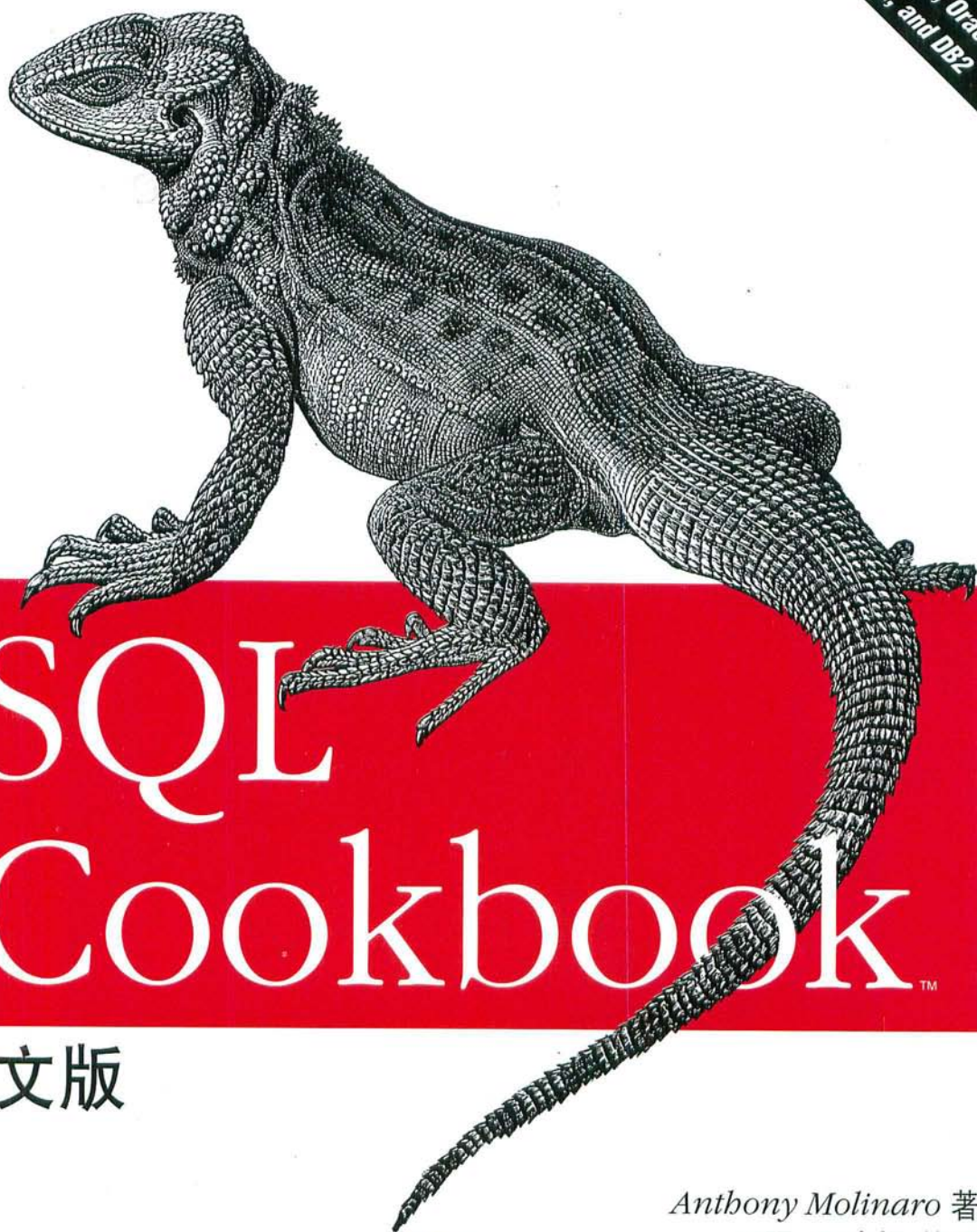


SQL Cookbook

涵盖 SQL Server,  
PostgreSQL, Oracle,  
MySQL, and DB2



# SQL Cookbook™

中文版

O'REILLY®



Anthony Molinaro 著  
王强 王晓娟 等译

清华大学出版社

---

# SQL Cookbook™

中文版

---

# SQL Cookbook™

中文版

*Anthony Molinaro* 著

王强 王晓娟 等译

O'REILLY®

*Beijing • Cambridge • Farnham • Köln • Paris • Sebastopol • Taipei • Tokyo*

O'Reilly Media, Inc. 授权清华大学出版社出版

清华大学出版社

[www.TopSage.com](http://www.TopSage.com)

Copyright ©2005 by O'Reilly Media, Inc.

Authorized Simplified Chinese translation edition, by O'Reilly Media, Inc., is published by Tsinghua University Press, 2007. Authorized translation of the original English edition, 2005 O'Reilly Media, Inc., the owner of all rights to publish and sell the same.

All rights reserved including the rights of reproduction in whole or in part in any form.

本书之英文原版由 O'Reilly Media, Inc. 于 2005 年出版。

本中文简体翻译版由 O'Reilly Media, Inc. 授权清华大学出版社于 2007 年出版。此翻译版的出版和销售得到出版权和销售权的所有者——O'Reilly Media, Inc. 的许可。

版权所有，未经书面许可，本书的任何部分和全部不得以任何形式复制。

北京市版权局著作权合同登记

图字：01-2006-7118 号

本书封面贴有清华大学出版社防伪标签，无标签者不得销售。

版权所有，侵权必究。侵权举报电话：010-62782989 13501256678 13801310933

## 图书在版编目 (CIP) 数据

SQL Cookbook™ 中文版 / (美) 莫利纳罗 (Molinaro, A.) 著; 王强、王晓娟等译.

—北京: 清华大学出版社, 2007. 10

书名原文: SQL Cookbook

ISBN 978-7-302-15493-8

I. S… II. ①莫… ②王… ③王… III. 关系数据库—数据库管理系统, SQL IV. TP311.138

中国版本图书馆 CIP 数据核字 (2007) 第 090296 号

责任编辑: 龙啟铭

封面设计: Karen Montgomery, 张 健

责任校对: 张 剑

责任印制: 王秀菊

出版发行: 清华大学出版社 地 址: 北京清华大学学研大厦 A 座

<http://www.tup.com.cn> 邮 编: 100084

[c-service@tup.tsinghua.edu.cn](mailto:c-service@tup.tsinghua.edu.cn)

社 总 机: 010-62770175 邮购热线: 010-62786544

投稿咨询: 010-62772015 客户服务: 010-62776969

印 刷 者: 清华大学印刷厂

装 订 者: 三河市溧源装订厂

经 销: 全国新华书店

开 本: 178 毫米×233 毫米 32.5 印张 字数: 706 千字

版 次: 2007 年 10 月第 1 版 印 次: 2007 年 10 月第 1 次印刷

印 数: 1~3000 册

定 价: 65.00 元(册)

本书如存在文字不清、漏印、缺页、倒页、脱页等印装质量问题, 请与清华大学出版社出版部  
联系调换。联系电话: (010)62770177 转 3103 产品编号: 023631-01

[www.TopSage.com](http://www.TopSage.com)



## O'Reilly Media, Inc. 介绍

为了满足读者对网络和软件技术知识的迫切需求，世界著名计算机图书出版机构 O'Reilly Media, Inc. 授权清华大学出版社，翻译出版一批该公司久负盛名的英文经典技术专著。

O'Reilly Media, Inc. 是世界上在 Unix、X、Internet 和其他开放系统图书领域具有领导地位的出版公司，同时也是联机出版的先锋。

从最畅销的 *The Whole Internet User's Guide & Catalog*（被纽约公共图书馆评为 20 世纪最重要的 50 本书之一）到 GNN（最早的 Internet 门户和商业网站），再到 WebSite（第一个桌面 PC 的 Web 服务器软件），O'Reilly Media, Inc. 一直处于 Internet 发展的最前沿。

许多书店的反馈表明，O'Reilly Media, Inc. 是最稳定的计算机图书出版商——每一本书都一版再版。与大多数计算机图书出版商相比，O'Reilly Media, Inc. 具有深厚的计算机专业背景，这使得 O'Reilly Media, Inc. 形成了一个非常不同于其他出版商的出版方针。O'Reilly Media, Inc. 所有的编辑人员以前都是程序员，或者是顶尖级的技术专家。O'Reilly Media, Inc. 还有许多固定的作者群体——他们本身是相关领域的技术专家、咨询专家，而现在编写著作，O'Reilly Media, Inc. 依靠他们及时地推出图书。因为 O'Reilly Media, Inc. 紧密地与计算机业界联系着，所以 O'Reilly Media, Inc. 知道市场上真正需要什么图书。

# 译者序

SQL 对大多数业内人士来说并不陌生，大凡编写程序的人大多用过 SQL。译者使用 SQL 也有十多年了，也算小有心得；然而，这本书的许多内容还是给我一种豁然开朗的感觉，翻译的过程真就是一个学习的过程。

这本书有以下几方面的特点：

实用。这不是一本关于 SQL 的教程，而是针对实际应用的需求提出了一百多个普遍性问题的解决方案，其中大部分都是从作者的日常实践中提炼出来的。有一定经验的 SQL 开发人员会发现，对其中大多数问题自己都曾经有过类似的需求，将这里的解决方案跟自己的做法对比一下一定会有所启发；对 SQL 新手而言，有了这本书就可以避免前人的探索过程，因为巨人已经给你准备好了肩膀。当然，实用并不是说其中的代码拿来就可以直接用，关键是其中的方法、技巧，有时书中的问题看似跟你的实际需要毫不相干，但其解决方案也有可能让你产生灵感，找到解决问题的途径。

富有创造性。从很多问题的解决方案中可以看出作者的创造性。例如求累乘积，译者一看到标题就感到疑惑，因为 SQL 中并没有类似 SUM 这样求乘积的聚集函数，译者过去碰到这样的需求都是用 SQL 把相关数据取出来，然后用其他语言老老实实在地将各行的数据一个个乘起来。看看书中的解决方案，原来道理很简单，初中就学过，但恐怕不是每个人都会想到的（也许是译者太孤陋寡闻吧）。如果能从书中学到点创造性思维方式，那收获就太大了。

分析深入。该书每个问题的解决方案之后都有讨论部分。讨论中分析了 SQL 中各部分的作用，得到什么结果，为什么会有这样的结果，很多地方揭示了隐藏在 SQL 语句背后的实质，例如各子句的处理顺序。译者曾经不止一次有过这样的经历：测试中发现 SQL 查询结果总是跟预期相左，怎么也看不出所以然，不得已只好绕道，使用其他方法。如果译者早读过这本书，情况恐怕就不一样了。另外，新手看到比较长的 SQL 语句往往会心里发毛，这本书的讨论中（特别是后面几章）会把复杂的 SQL 分解开，从最简单的形式一步步引向最终解决方案。相信这样的分析方法一定会让读者有所裨益的。

针对多种平台。本书中为每个问题提供了针对 5 种 RDBMS 的解决方案（第 14 章除外），其中也有许多问题是多个平台使用同一个方案，因为这些平台的有关特性相同或十分相似。当然，跟多平台相对的就是可移植性问题，这主要取决于各人偏好，当然，更主要是实际应用的需要，不可一概而论。

本书由王强、王晓娟翻译，樊庆红统校，参加翻译的还有李强、赵东辉、李伟、郭涛、高磊、王振营、冯哲、韩毅、马以辉、李腾、周云、董武、郑晓蕊、陈占军、倪泳智、黄虹、吕巧珍、裘蕾、金颖、韩毅、王嘉佳、吴建伟、宋雁、邓卫、何晓刚、段涛、马丽娟、郭翔、朱晓林、陈磊、李建锋、刘延军、刘子瑛、徐英武、魏宇、赵远锋、邓蛟龙、樊旭平、刘延军、唐玮、魏宇、吕巧珍等人。

最后，祝广大读者从本书中挖掘出更多的宝藏。

# 目录

前言 .....	1
第 1 章 检索记录 .....	15
1.1 从表中检索所有行和列 .....	15
1.2 从表中检索部分行 .....	15
1.3 查找满足多个条件的行 .....	16
1.4 从表中检索部分列 .....	17
1.5 为列取有意义的名称 .....	17
1.6 在 WHERE 子句中引用取别名的列 .....	18
1.7 连接列值 .....	19
1.8 在 SELECT 语句中使用条件逻辑 .....	20
1.9 限制返回的行数 .....	20
1.10 从表中随机返回 n 条记录 .....	22
1.11 查找空值 .....	23
1.12 将空值转换为实际值 .....	24
1.13 按模式搜索 .....	24

<b>第 2 章 查询结果排序 .....</b>	<b>26</b>
2.1 以指定的次序返回查询结果 .....	26
2.2 按多个字段排序 .....	27
2.3 按子串排序 .....	28
2.4 对字母数字混合的数据排序 .....	28
2.5 处理排序空值 .....	31
2.6 根据数据项的键排序 .....	36
<b>第 3 章 操作多个表 .....</b>	<b>37</b>
3.1 记录集的叠加 .....	37
3.2 组合相关的行 .....	38
3.3 在两个表中查找共同行 .....	40
3.4 从一个表中查找另一个表没有的值 .....	41
3.5 在一个表中查找与其他表不匹配的记录 .....	44
3.6 向查询中增加联接而不影响其他联接 .....	46
3.7 检测两个表中是否有相同的数据 .....	48
3.8 识别和消除笛卡儿积 .....	53
3.9 聚集与联接 .....	55
3.10 聚集与外联接 .....	58
3.11 从多个表中返回丢失的数据 .....	61
3.12 在运算和比较时使用 NULL 值 .....	64
<b>第 4 章 插入、更新与删除 .....</b>	<b>65</b>
4.1 插入新记录 .....	65
4.2 插入默认值 .....	66
4.3 使用 NULL 代替默认值 .....	67
4.4 从一个表向另外的表中复制行 .....	68
4.5 复制表定义 .....	68
4.6 一次向多个表中插入记录 .....	69
4.7 阻止对某几列插入 .....	71

4.8	在表中编辑记录 .....	72
4.9	当相应行存在时更新 .....	73
4.10	用其他表中的值更新 .....	74
4.11	合并记录 .....	77
4.12	从表中删除所有记录 .....	78
4.13	删除指定记录 .....	78
4.14	删除单个记录 .....	79
4.15	删除违反参照完整性的记录 .....	79
4.16	删除重复记录 .....	80
4.17	删除从其他表引用的记录 .....	82

## 第 5 章 元数据查询 ..... 83

5.1	列出模式中的表 .....	83
5.2	列出表的列 .....	84
5.3	列出表的索引列 .....	85
5.4	列出表约束 .....	86
5.5	列出没有相应索引的外键 .....	87
5.6	使用 SQL 来生成 SQL .....	90
5.7	在 Oracle 中描述数据字典视图 .....	91

## 第 6 章 使用字符串 ..... 93

6.1	遍历字符串 .....	93
6.2	字符串文字中包含引号 .....	95
6.3	计算字符在字符串中出现的次数 .....	96
6.4	从字符串中删除不需要的字符 .....	97
6.5	将字符和数字数据分离 .....	98
6.6	判别字符串是不是字母数字型的 .....	102
6.7	提取姓名的大写首字母缩写 .....	106
6.8	按字符串中的部分内容排序 .....	109
6.9	按字符串中的数值排序 .....	110

6.10	根据表中的行创建一个分隔列表 .....	115
6.11	将分隔数据转换为多值 IN 列表 .....	120
6.12	按字母顺序排列字符串 .....	125
6.13	判别可作为数值的字符串 .....	130
6.14	提取第 n 个分隔的子串 .....	135
6.15	分解 IP 地址 .....	141
 <b>第 7 章 使用数字 .....</b>		<b>143</b>
7.1	计算平均值 .....	143
7.2	求某列中的最小 / 最大值 .....	145
7.3	对某列的值求和 .....	146
7.4	求一个表的行数 .....	147
7.5	求某列值的个数 .....	149
7.6	生成累计和 .....	150
7.7	生成累乘积 .....	152
7.8	计算累计差 .....	155
7.9	计算模式 .....	156
7.10	计算中间值 .....	158
7.11	求总和的百分比 .....	161
7.12	对可空列作聚集 .....	163
7.13	计算不包含最大值和最小值的均值 .....	164
7.14	把字母数字串转换为数值 .....	166
7.15	更改累计和中的值 .....	168
 <b>第 8 章 日期运算 .....</b>		<b>170</b>
8.1	加减日、月、年 .....	170
8.2	计算两个日期之间的天数 .....	172
8.3	确定两个日期之间的工作日数目 .....	174
8.4	确定两个日期之间的月份数或年数 .....	178
8.5	确定两个日期之间的秒、分、小时数 .....	180



8.6	计算一年中周内各日期的次数 .....	181
8.7	确定当前记录和下一条记录之间相差的天数 .....	191
<b>第 9 章 日期操作 .....</b>		<b>196</b>
9.1	确定一年是否为闰年 .....	196
9.2	确定一年内的天数 .....	202
9.3	从日期中提取时间的各部分 .....	205
9.4	确定某个月的第一天和最后一天 .....	206
9.5	确定一年内属于周内某一天的所有日期 .....	209
9.6	确定某月内第一个和最后一个“周内某天”的日期 .....	214
9.7	创建日历 .....	220
9.8	列出一年中每个季度的开始日期和结束日期 .....	235
9.9	确定某个给定季度的开始日期和结束日期 .....	239
9.10	填充丢失的日期 .....	245
9.11	按照给定的时间单位进行查找 .....	253
9.12	使用日期的特殊部分比较记录 .....	254
9.13	识别重叠的日期范围 .....	257
<b>第 10 章 范围处理 .....</b>		<b>262</b>
10.1	定位连续值的范围 .....	262
10.2	查找同一组或分区中行之间的差 .....	266
10.3	定位连续值范围的开始点和结束点 .....	274
10.4	补充范围内丢失的值 .....	277
10.5	生成连续数字值 .....	281
<b>第 11 章 高级查找 .....</b>		<b>285</b>
11.1	给结果集分页 .....	285
11.2	跳过表中 n 行 .....	288
11.3	在外联接中用 OR 逻辑 .....	290
11.4	确定哪些行是彼此互换的 .....	292

11.5	选择前 n 个记录 .....	294
11.6	找到包含最大值和最小值的记录 .....	295
11.7	存取“未来”行 .....	296
11.8	轮换行值 .....	299
11.9	给结果分等级 .....	302
11.10	抑制重复 .....	303
11.11	找到骑士值 .....	305
11.12	生成简单的预测 .....	311

## 第 12 章 报表和数据仓库运算..... 318

12.1	将结果集转置为一行 .....	318
12.2	把结果集转置为多行 .....	320
12.3	反向转置结果集 .....	326
12.4	将结果集反向转置为一列 .....	328
12.5	抑制结果集中的重复值 .....	331
12.6	转置结果集以利于跨行计算 .....	333
12.7	创建固定大小的数据桶 .....	335
12.8	创建预定数目的桶 .....	338
12.9	创建横向直方图 .....	342
12.10	创建纵向直方图 .....	344
12.11	返回未包含在 GROUP BY 中的列 .....	346
12.12	计算简单的小计 .....	351
12.13	计算所有表达式组合的小计 .....	354
12.14	判别非小计的行 .....	362
12.15	使用 Case 表达式给行做标记 .....	363
12.16	创建稀疏矩阵 .....	365
12.17	按时间单位给行分组 .....	366
12.18	对不同组 / 分区同时实现聚集 .....	369
12.19	对移动范围的值进行聚集 .....	371
12.20	转置带小计的结果集 .....	377

<b>第 13 章 分层查询 .....</b>	<b>381</b>
13.1 表示父－子关系 .....	382
13.2 表示子－父－祖父关系 .....	385
13.3 创建表的分层视图 .....	389
13.4 为给定父行找到所有子行 .....	396
13.5 确定哪些行是叶节点、分支节点及根节点 .....	399
 <b>第 14 章 若干另类目标 .....</b>	 <b>406</b>
14.1 使用 SQL Server 的 PIVOT 运算符创建交叉表报表 .....	406
14.2 使用 SQL Server 的 UNPIVOT 运算符反转置交叉表报表 .....	408
14.3 使用 Oracle 的 MODEL 子句转换结果集 .....	409
14.4 从不固定位置提取字符串的元素 .....	413
14.5 求一年包含的天数 (Oracle 的另一种解决方案) .....	415
14.6 搜索字母数字混合的字符串 .....	416
14.7 使用 Oracle 把整数转换为二进制数 .....	418
14.8 转置已分等级的结果集 .....	420
14.9 给两次转置的结果集增加列头 .....	423
14.10 在 Oracle 中把标量子查询转换为复合子查询 .....	432
14.11 把连续数据分解为行 .....	434
14.12 计算相对于总数的百分数 .....	438
14.13 从 Oracle 创建 CSV 格式输出 .....	439
14.14 找到与模式不匹配的文本 (Oracle) .....	444
14.15 用内联视图转换数据 .....	446
14.16 测试一个组内是否存在某个值 .....	447
 <b>附录 A 窗口函数补充 .....</b>	 <b>451</b>
 <b>附录 B 回顾 Rozenshtein .....</b>	 <b>473</b>

# 前言

SQL 是计算机世界的语言，在用关系数据库开发报表时，将数据放入数据库以及从数据库中取出来，都需要 SQL 的知识。很多人以一种马马虎虎的态度在使用 SQL，根本没有意识到自己掌握着多么强大的武器。本书的目的是打开读者的视野，看看 SQL 究竟能干什么，以改变这种状况。

本书是一本指南，其中包含了一系列 SQL 的常用问题以及它们的解决方案，希望能对读者的日常工作有所帮助。本书将相关主题的小节归成章，如果读者遇到不能解决的 SQL 新问题，可以先找到最可能适用的章，浏览其中各小节的标题，希望读者能从中找到解决方案，至少可以找到点灵感。

在这本书中有 150 多个小节，这还仅仅是 SQL 所能做的事情的一鳞半爪。解决日常编程问题的解决方案的数量仅取决于需要解决的问题的数量，本书没有覆盖所有问题，事实上也不可能覆盖，然而从中可以找到许多共同的问题及其解决方案，这些解决方案中用到许多技巧，读者学到这些技巧就可以将它们扩展并应用到本书不可能覆盖的其他新问题上。

---

**注意：**我和出版商一直在搜集适合指南的新 SQL 方案，如果您有对某问题的好的（或聪明的）解决方案，能否与大家共享？能否发给我们，以便用于本书的下一个版本。有关我们的联系信息请参阅“注释与问题”。

---

## 为何要写本书

查询，查询，还是查询。项目开始时我并没有如此强烈的愿望要将“SQL 指南”写成本“查询指南”。我的目标是写这么一本书，其中包含从简单到复杂的查询，使读者能掌握查询中使用的技巧，并用来解决他们自己的实际业务问题。我希望能将自己职业生涯

中所积累的 SQL 编程技巧传递给读者，使他们能够看到、能够从中学到，并最终可以加以改进；通过这么一个循环，我们双方都会获益。从数据库中检索数据看似一件容易的事，然而，在 IT 世界里，尽可能高效地检索数据至关重要，高效检索数据的技巧应该与大家分享，这样大家都会很高效，并互相帮助、共同提高。

想一想数学家 Georg Cantor 的杰出贡献，是他首先认识到研究元素集合（研究集合本身，而不是其中的元素）的巨大利益。起初，Cantor 的工作并没有得到他许多同伴的认同，尽管如此，随着时间的推移，不仅他的工作得到了认同，而且现在人们已经把集合论视为数学的基础。然而更重要的是，集合论取得今天的成就并非 Cantor 一个人的功劳，通过分享他的思想，其他人，如 Ernst Zermelo、Gottlob Frege、Abraham Fraenkel、Thoralf Skolem、Kurt G<sub>del</sub> 以及 John von Neumann 等人，对集合理论作了进一步的发展和提高。这样的分享不仅使每个人可以更好地理解集合论，而且使结合论本身比最初的构想更趋完善。

## 本书的目标

毫无疑问，本书的目标是让读者看到，SQL 能够做多少一般认为是 SQL 问题范围之外的事情。在过去的 10 年间，SQL 走过了很长的路，许多过去只能用 C 和 JAVA 等过程化语言解决的典型问题现在都可以直接用 SQL 解决了，但是很多开发人员并没有意识到这一事实。本书就是要帮助大家认识到这一点。

现在，在对我刚才的话产生误解之前我先要申明：我是“如果没坏，就别去修它”这一教义的忠实信徒。例如，假如你有一个特定的业务问题要解决，目前只用 SQL 检索数据，而其他复杂的业务逻辑由其他语言完成，如果代码没有问题，而且性能也过得去，那么，谢天谢地。我绝对无意建议你扔掉以前的代码重新寻求完全 SQL 的解决方案；我只是请你敞开思想，认识到 1995 年编程用的 SQL 跟 2005 年用的不是一回事，今天的 SQL 能做的事要多得多。

## 本书的读者

本书在这一点上很独特：其目标读者非常广泛，而其内容却没有做折衷。想一想，其中既有简单方案，又有复杂方案；而且当一个问题没有统一方案时会提供 5 个不同数据库厂商的解决方案。本书的目标读者确实很广泛：

### SQL 新手

你可能刚买了本 SQL 的书开始学习，也可能是初修数据库课程的学生，你可能希望通过一些富有挑战性的现实中的例子来拓展自己的知识面。你以前可能看到过用查询魔术般地将行转换成列，或者将连续的字符串解析成结果集，本书中的各节将向你介绍这些看似不可能的查询中所使用的技巧。

### 非 SQL 程序员

你的背景可能是其他语言，而目前的工作要求你去维护别人编写的复杂的 SQL，你觉得犹如置身火中。本书中的各节，尤其是后面几章，会将复杂的查询分解开，带你一步步“潇洒走一回”，帮你理解可能会接手的复杂代码。

### SQL 高手

对中级 SQL 开发人员而言，本书就是“彩虹那端的黄金”（对，可能说得太过了；请原谅一个作者对自己理念的醉心）。特别是，如果你编写 SQL 代码时间不短了，但还没有用上窗口函数，那么本书对你就是一顿大餐。例如，需要用临时表存放中间结果集的日子已经过去了，利用窗口函数可以用一个查询就解决问题！请允许我重申一下：我无意向任何有经验的同仁灌输我的思想；如果你还没有理解 SQL 的新增功能的话，请将本书作为更新技能的途径。

### SQL 专家

毫无疑问，你以前已经看到过这样的方案了，而且自己可能有所改进，那么本书对你还有什么用呢？在你的整个职业生涯中，你可能是某个平台的 SQL 专家，比如，SQL Server，现在想学 Oracle；你可能只用过 MySQL，并且想知道同一问题在 PostgreSQL 中的解决方案是怎么样的。本书覆盖了 5 种不同的关系数据库系统（RDBMS），并一步步展示这些解决方案。这是你拓展知识基础的好机会。

## 如何使用本书

务必请通读本前言，如果直接进入各小节，就会错失一些背景信息。“平台和版本”一节交待了本书所覆盖的 RDBMS；要特别注意一下“本书使用的表”一节，这样就会熟悉大部分小节中的示例表，“本书的约定”一节中包含了一些重要的代码和字体约定。所有这些小节都是本章的内容。

记住这是一本指南，是一系列代码示例，用于指导解决可能遇到的类似（或相同）的问题。本书不是用来学习 SQL 的，至少不是从头开始，它可以用作 SQL 教程的补充，但不能代替教程。此外，以下提示会有助于更有效地使用本书：

- 本书中使用了各厂商的专用函数，Jonathan Gennick 的 SQL Pocket Guide 一书中都有介绍，如果你对本书各小节中有些函数还不了解的话，有这么一本书在手头是很方便的。
- 如果你从来没有使用过窗口函数，或者对使用 GROUP BY 的查询有问题，请先看附录 A，其中会给出 SQL 中组的定义并作证明；更重要的是，它介绍了窗口函数工作方式的基本思路，窗口函数是过去十年中 SQL 最重要的发展。
- 利用常识。要知道，不可能写一本包罗所有问题解决方案的书，然而，可以将本书中的方案作为模板或指导思想，让自己学到其中的技巧，解决自己的问题。有人可



能会说：“好是好，就是这些方案只能用于书上那个数据集，而我的数据集不一样，所以不能用”。一点也不意外。如果是这样的话，先设法找出书上的数据跟你自己的数据之间的共同点，将书中的查询分解成最简单的形式，然后逐步增加其复杂度。所有的查询都包含 `SELECT ... FROM ...`，这是它们最简单的形式，所有查询都一样；在增加复杂度的过程中，可以每次增加一个函数，或每次增加一个联接，这样不但有助于理解查询结构对结果集的影响，而且可以看出书中的方案跟自己的实际需要之间差别何在。然后，就可以按照自己的数据集做相应的修改。

- 测试，测试，再测试。毫无疑问，实际使用的表比书中所举的 14 行的 EMP 表要大得多，所以要用自己的数据集对解决方案做测试，锱铢必较地致力提高性能。我不可能知道你的表是什么样的，不知道它们如何索引，也不知道模式中的关系，所以，除非你完全理解了这些技巧以及它们在自己数据上的性能，请不要盲目在产品代码中使用这些技巧。
- 要勇于试验，要有创新精神，你完全可以使用其他技巧。我在本书中用了各厂商特定的函数，通常其他还有许多其他函数完成同样的功能。你可以随意对本书给出的方案进行修改。
- 新并不一定意味着好。如果你目前没有使用 SQL 语言的一些新特性（比如窗口函数），这并不意味着你的代码一定不高效。很多情况下传统 SQL 解决方案跟新方案不相上下，有时甚至更好。请一定要记住这一点，尤其是在附录 B“回顾 Rozenshtein”中。读完本书，你不应该觉得要将现有所有代码重写，而是要认识到，现在 SQL 已经有更多非常高效的特性，这些东西 10 年前是没有的，很值得花点时间去学习。
- 不要被吓倒。如果一个解决方案看起来不可能理解，不要害怕。我竭尽全力，不仅将查询分解，从最简单的形式开始，而且还给出了每一部分的中间结果。你可能不能马上看清全局，但只要跟随着讨论的思路，就可以看到查询建立起来的过程集及中间结果，而且会发现，即使是看起来非常费解的查询其实也不难掌握。
- 必要时采取防范措施。在不导致误解或歧义的前提下，为了尽可能保持书中查询的简洁，我去掉了许多“防范措施”。例如，对计算员工工资累加和的查询，情况可能会是这样的：工资列定义为 `VARCHAR` 类型，而且（不幸的是）其中有一个字段中混合了文字，你会发现本书计算累加和的解决方案中并没有检查这种情形（这样，因为 `SUM` 函数不知道如何处理字符，查询就会失败），如果你的“数据”（更确切地说是“问题”）是这样的，就需要加些代码来处理，或者（希望可以）修正数据中的错误，因为在设计解决方案时并没有考虑同一列中既包含字母又包含数字。本书的思想是专注于技巧；一旦你理解了其中的技巧后，锦上添花并非难事。
- 重复是关键。掌握本书各解决方案最好的方法是坐下来写代码，一旦开始编写代码，阅读就会仔细，而实际上编写代码本身作用更大。要理解为什么要采用某种特定的方式，这就需要阅读；而只有编写代码，自己才能创建查询。



请注意，本书中许多例子是杜撰的，但问题不是杜撰的，它们是真实的。然而，我一直在用包含员工信息的很少的几个表举例，这样做是为了使读者能够熟悉示例数据，以便能够专注于各节阐明技巧。这些技巧很有用，我和我的同事日常都在使用，相信你会用的。

## 本书中缺少什么

由于时间和容量的限制，一本书中不可能包含各种方案来解决可能遇到的所有SQL问题。例如以下就是本书没有覆盖的内容：

### 数据定义

本书没有覆盖SQL中像创建索引、增加约束以及装载数据等方面的内容。这类任务一般都包含厂商非常专用的语法，所以最好还是去查相应厂商的手册。此外，这类问题也不能代表某种“硬”问题类型，用户买本书就可以解决。然而，第4章却提供了插入、删除、更新等共同问题的解决方案。

### XML

我极不愿意将XML的有关解决方案加到这本关于SQL的书。将XML文档存在关系数据库中正越来越普及，每种RDBMS都各自做了扩展，并有专门的工具来检索、操纵这类数据。XML操纵通常涉及过程性的代码，因而不在于本书范围之内；近来的发展，如XQUERY等，跟SQL是完全不同的主题，应有其他专门书籍介绍。

### SQL面向对象的扩展

在更适合处理对象的语言出现之前，我还是强烈反对在关系数据库中使用面向对象的特性和设计。目前，部分厂商提供的面向对象的特性更适合用在过程性的语言中，而不是像SQL这种解决面向集合问题的语言。

### 理论问题的争论

本书中不会找到有关“SQL是不是关系型的”或“NULL值该不该存在”等方面的争论。理论问题的争论有它们自己的场所，不应该出现在以现实问题提供SQL解决方案为中心的书中。要解决现实问题，只需使用当时可用的工具，你只能利用现有的东西，而不是你希望有的东西。

---

注意：如果你想更多地了解理论，Chris Date的任何“Relational Database Writings”都可以引你入门；你也可以使用他最近的新作，Database in Depth (O'Reilly)。

---

### 供应商策略

书中提供了5种不同RDBMS的解决方案，想知道哪个厂商的方案“最好”或“最快”是很自然的想法。每家厂商都很乐意提供这方面的丰富信息，并以此表明他们的产品是“最好的”。这里，我无意做这方面的工作。

### ANSI 策略

很多书远离不同厂商提供的专用函数，而本书则充分使用了这类专用函数。我不愿意仅仅为了可移植性而去编写那种令人费解并且性能低下的SQL代码，我工作过的工作环境从来没有禁止使用厂商专有的扩展功能。既然你为这些特性买了单，为什么不充分利用呢？

厂商的扩展自有它存在的理由，而且大多数情况下性能和可读性比用标准SQL更好。如果你喜欢只使用ANSI解决方案，那没问题，我前面说过，我并不是要让你颠覆以前的代码。如果你现有的代码都是纯ANSI的，并且运转良好，那棒极了。归根到底，我们都去上班，都要付账，都希望在合理的时间回家，并享受其余的时光。因此，我的意思并不是说只用ANSI就不对，但是，我要说清楚，如果要找纯ANSI的解决方案，那么得另寻他路。

### 落伍策略

本书使用了到编写时为止最新的特性，如果你使用的是本书所覆盖的RDBMS较老的版本，那么大部分解决方案可能不能为你所用。技术不会停滞不前，你也一样。如果你需要老版本解决方案，很多前几年出版的书中有大量使用这些老版本RDBMS的例子。

## 本书的结构

本书划分为14章和两个附录：

- 第1章，检索记录，介绍了一些很简单的查询，例如，如何使用WHERE子句限制结果集的行数，给结果集的列提供别名，用内联视图引用别名列，使用简单的条件逻辑，限制查询返回的行数，返回随即记录，以及查找NULL值等。其中大部分都很简单，但是以后复杂的查询中可能会用到其中一部分，所以如果你是新学SQL，或者对上面这些例子不熟悉的话，最好先阅读本章。
- 第2章，查询结果排序，介绍了有关对查询结果排序的方案。本章介绍了ORDER BY子句，并用它来对查询结果排序。其中的例子复杂度逐渐增加，从简单的单列排序，到按子串排序，再到按条件表达式排序。
- 第3章，操作多个表，介绍了把多个表的数据组合起来的方案。如果你是SQL的新手，或者对联接觉得有点生疏的话，强烈建议在阅读第5章及其后各章之前先阅读本章。表联接就是SQL的全部，要成功就必须理解联接。本章的例子有，进行内联接和外联接，识别笛卡儿积，基本的集合运算，按聚集函数联接的结果等。
- 第4章，插入、更新与删除，分别介绍了对数据的插入、更新以及删除操作。大部分例子都很直接（甚至可以说呆板），然而，像从一个表向另一个表插入，更新中使用关联子查询，理解NULL的影响以及有关多表插入、MERGE命令等新个性的知识等都是非常有用的工具。

- 第5章，元数据查询，介绍了获取数据库中元数据的方案。找出模式中的索引、约束以及表定义等信息通常都非常有用，本章的这些简单解决方案可以获得模式的有关信息。此外，这里也给出了动态SQL的例子，也就是用SQL产生SQL。
- 第6章，使用字符串，介绍了操纵字符串的方案。SQL解析字符串的能力并不知名，然而，发挥点创造力（通常会用到笛卡儿积）加上大量厂商专用函数，就很能做点事情。从这章开始，本书就开始有意思了。其中比较有意思的例子有，计算字符在字符串中出现的次数，根据表中的行创建一个分隔列表，将分隔列表和字符串转换成分行，以及将字母数字串中的字母和数字分离等。
- 第7章，使用数字，介绍了常见的数值处理。这一章的例子都很常见，并且可以从中学到在涉及移动计算和聚集方面，窗口函数多么方便。该章的例子有，累加和，查找平均数、中间数以及模式，计算百分比，以及进行聚集时对NULL值的处理等。
- 第8章，日期运算，是处理日期的两章中的第1章。对日常任务而言，对简单日期类型的操作非常重要。这一章的例子有：两个日期之间相差的天数，按不同时间单位（日、月、年等）求两个日期间的差，计数一个月中的天数等。
- 第9章，日期操作，是处理日期的两章中的第2章。这一章中可以找到有关日常会遇到的一些最常用的日期操作的解决方案。例如，求一年的天数，查找闰年，求一月中的第一天和最后一天，创建日历，以及在一个日期范围内填补缺失的数据等。
- 第10章，范围处理，介绍了识别范围和创建范围有关的方案。例如，自动产生行序列，以及在一个范围内填补缺失的数值，定位连续值范围的开始点和结束点，以及定位相邻值等。
- 第11章，高级搜索，介绍了日常开发中非常重要然而却有时被忽略的一些问题的方案。这一章的方案不见得就比其他的难，但是我看到许多开发人员解决这些问题时效率十分低下。这一章的例子有，求奇数值，按页滚动数据集，跳过表中的行，查找互换列值的行，选择前n条记录以及给结果分等级等。
- 第12章，报表和数据仓库操作，介绍了一般用于数据仓库或产生复杂报表的查询。这一章应是本书最主要的一章，我最初的打算中就有这部分内容。这一章的例子有，将行转换成列以及反向转换（交叉表报表），创建数据桶或组，创建柱状图，计算简单和复杂的小计，以及根据给定时间单位分组行等。
- 第13章，层次查询，介绍了层次型的解决方案。不管数据模型如何建立，总有时会将数据组织成表示树形或父-子关系的结构，这一章的解决方案就是要完成这类任务。用传统SQL创建树状结构非常麻烦，因此这一章中厂商专用的函数作用很大。本章的例子有，表示父-子关系，从根到叶遍历层次结构，以及层次上卷等。
- 第14章，若干另类目标，包括一些似乎不大适合其他问题领域然而有十分有趣、有用的解决方案。这一章跟其他各章不同，专注于一些厂商专用的解决方案，这是唯

一的一章，其中的解决方案只强调一个厂商，这里有双重原因：其一，这一章的意图就是提供一些乐趣；其二，这里的解决方案只强调一家厂商的专用函数，因为其他 RDBMS 中没有等价的特性（例如 SQL Server 的 PIVOT/UNPIVOT 操作符和 Oracle 的 MODEL 子句）。然而，有些情况下倒是可以将这里的解决方案稍加修改，应用于本书没有涉及的其他平台。

- 附录 A，窗口函数回顾，回顾了窗口函数，并对 SQL 的组作了深入的讨论。对大多数人而言，窗口函数是新的，所以用这个附录作为简略的指南还是恰当的。另外，在我的经历中，我发现在查询中使用 GROUP BY 使很多人觉得很迷惑。这一章给了 SQL 组一个精确的定义，并用各种各样的查询来验证这个定义；本章然后转入讨论 NULL 在组、聚集和分区中的影响，最后对更晦涩，然而特别强大的窗口函数 OVER 子句（也就是窗口和框架子句）的语法作了讨论。
- 附录 B，回顾 Rozenshtein，是献给 David Rozenshtein 的礼物，我在 SQL 方面的成功完全要归功于他。Rozenshtein 的书 *Essence of SQL* 是除课堂需要外我买的第一本关于 SQL 的书，就是从这开始我学会了如何用 SQL 的方式思考；直到今天，我仍把许多对 SQL 工作机理的理解归结于 David 的书。这本书跟我所看过的其他 SQL 的书确实不一样，我很感激我按自己的意愿买的第一本书就是那本。附录 B 关注的是 *Essence of SQL* 中提出的几个查询，并为这些查询提供了窗口函数的版本（写 *Essence of SQL* 时还没有窗口函数）。

## 平台与版本

SQL 在向前发展，厂商一直不断地向其产品中加入新特性和功能，因而你应该预先知道本书所面向的平台版本。

- DB2 v.8
- Oracle Database 10g（部分方案除外，这些解决放按也能用于 Oracle8i Database 和 Oracle9i Database）
- PostgreSQL 8
- SQL Server 2005
- MySQL 5

## 本书所使用的表

本书的大部分例子使用了两个表：EMP 和 DEPT。EMP 是一个只有 14 行的简单表，其中只有数值、字符串和日期类型的字段。这些表在比较老的数据库的书中都出现过，并且部门和员工之间一对多的关系也很容易理解。

既然说到示例表，我要提一下，在少数解决方案中用到3个表。我绝不会把示例数据弄得很别扭，使读者不太有机会将这些方案应用到现实中，有的书就是这么做的。

既然说到解决方案，我还要提一下，只要有可能，我会尽量提供使用与本书所涉及的5种RDBMS的通用解决方案，但通常这不大可能。尽管如此，在很多情况下有多种平台使用同一个解决方案，因为它们共同支持窗口函数，例如，Oracle和DB2经常使用同一方案。只要几个平台使用同一方案，或者它们的方案很相近，就会将它们放在一起讨论。

EMP和DEPT表的内容分别如下：

```
select * from emp;
```

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7369	SMITH	CLERK	7902	17-DEC-1980	800		20
7499	ALLEN	SALESMAN	7698	20-FEB-1981	1600	300	30
7521	WARD	SALESMAN	7698	22-FEB-1981	1250	500	30
7566	JONES	MANAGER	7839	02-APR-1981	2975		20
7654	MARTIN	SALESMAN	7698	28-SEP-1981	1250	1400	30
7698	BLAKE	MANAGER	7839	01-MAY-1981	2850		30
7782	CLARK	MANAGER	7839	09-JUN-1981	2450		10
7788	SCOTT	ANALYST	7566	09-DEC-1982	3000		20
7839	KING	PRESIDENT		17-NOV-1981	5000		10
7844	TURNER	SALESMAN	7698	08-SEP-1981	1500	0	30
7876	ADAMS	CLERK	7788	12-JAN-1983	1100		20
7900	JAMES	CLERK	7698	03-DEC-1981	950		30
7902	FORD	ANALYST	7566	03-DEC-1981	3000		20
7934	MILLER	CLERK	7782	23-JAN-1982	1300		10

```
select * from dept;
```

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON

此外，本书中还用到4个基干表：T1、T10、T100和T500。因为这些表这用于作为产生结果集的骨架，我觉得没必要给它们取更聪明的名字。这些表中“T”后面的数字表示相应表中的行数，其中的值从1开始递增。例如，表T1和T10的值分别如下：

```
select id from t1;
```

ID
1

```
select id from t10;
```

ID
1
2
3
4
5
6
7
8

有些厂商允许部分 SELECT 语句，例如，SELECT 语句可以没有 FROM 子句。我非常不喜欢这种形式，所以对只有一行的支持表 T1 进行 SELECT，而不使用部分查询。

其他表只有特定的方案和特定地章节专用，书中会在适当的时候介绍。

## 本书使用的约定

本书使用了一些排版和代码的约定，请花点时间熟悉它们，这样做会有助于对内容的理解。代码约定尤其重要，因为我不可能在书中的每一小节再重新交待，因此，将一些重要的约定列在这里。

### 排版约定

书中使用了下列排版约定：

#### 大写字母

用于表示正文中的 SQL 关键字

#### 小写字母

用于所有查询的代码示例。有些语言，如 C 和 JAVA，规定大部分关键字用小写字母，我发现这样比用大写字母可读性更好，因此所有查询都用小些字母。

#### 黑体内容

表示用户在示例中的交互输入内容。

### 编码约定

我喜欢在 SQL 语句中使用小写字母，不管是关键字还是用户标示符。例如：

```
select empno, ename
from emp;
```

你的偏好可能跟我不同，例如，很多人喜欢将 SQL 关键字大写。你可以使用自己喜欢的代码风格或按照项目要求去做。

尽管我在代码示例中使用小写字母，但正文中 SQL 关键字都使用大写字母，这样是为了强调这些项目不是普通文字。例如

前面的查询代码对 EMP 表的一次 SELECT。

尽管本书覆盖了 5 家不同的厂商，但我决定输出采用同一种格式：

```
EMPNO  ENAME
-----
7369   SMITH
```

```
7499 ALLEN
```

```
...
```

许多解决方案的FROM子句中用到了内联视图或子查询，ANSI SQL标准要求给这样的视图取别名（Oracle是唯一一个要求必须有别名的平台），因此我的解决方案中都给内联视图取了x、y等别名。例如：

```
select job, sal
  from (select job, max(sal) sal
        from emp
        group by job) x;
```

注意，X在最后一个右括号的后面，这样字母X就成了FROM子句中子查询的“表”名。尽管列别名是编写自含文档代码的有价值的工具，内联视图的别名（本书中绝大部分方案中）则只不过是形式主义。所以一般只给它们取个平凡的名字，如X、Y、Z、TMP1和TMP2等。如果我觉得有意义的别名有助于理解的话，我会那么做的。

你会注意到，解决方案中的SQL一般有编号，例如：

```
1 select ename
2   from emp
3  where deptno = 10
```

这些编号不是语法的一部分，使用编号只是为了在讨论部分方便引用。

## 使用代码示例

这本书是为了帮助读者做好工作，总体来说，你可以将本书的代码用于你的程序或文档。你不必联系O'Reilly以获得许可，除非你要复制相当大一部分的代码。例如，在程序中使用书中的几段代码不需要得到许可，销售或分发O'Reilly书中代码的CD-ROM需要得到许可；在回答问题时引用本书或引用示例代码不需要许可，将本书中相当数量的代码合并到你们自己的产品文档中需要或的许可。

我们非常感谢，但并不要求标明出处。“出处”通常包括书名、作者、出版商和ISBN，例如：SQL Cookbook, by Anthony Molinaro. Copyright 2006 O'Reilly Media, Inc., 0-596-00976-3。

如果你觉得对示例代码的使用不属于合理使用以上给出的许可情形，欢迎跟我们联系：[permissions@oreilly.com](mailto:permissions@oreilly.com)。

## 注释和问题

我们已经尽我们所能测试和校对过本书中的信息，但你可能还是会发现有些特性改变了，或者我们有错误。如果有这种情况，请写信到以下地址通知我们：



美国：

O'Reilly Media, Inc.  
1005 Gravenstein Highway North  
Sebastopol, CA 95472

中国：

100080 北京市海淀区知春路 49 号希格玛公寓 B 座 809 室  
奥莱理软件（北京）有限公司

本书的 Web 页上列出了勘误表、示例和任何额外的信息。可登录以下网址查询：

<http://www.oreilly.com/catalog/sqlckbk>  
<http://www.oreilly.com.cn/book.php?bn=978-7-302-15493-8>

如果想就本书的技术问题发表评论或咨询，请发邮件至：

[info@mail.oreilly.com.cn](mailto:info@mail.oreilly.com.cn)  
[bookquestions@oreilly.com](mailto:bookquestions@oreilly.com)

关于图书、会议、资源中心和 O'Reilly 网络的更多信息，请访问 O'Reilly 的 Web 站点：

<http://www.oreilly.com>  
<http://www.oreilly.com.cn>

## 致谢

如果没有很多人的支持这本书就不会面世。我要感谢我的母亲 Connie，我将这本书献给您。如果没有您的辛勤劳动和牺牲，就不可能有我的今天。妈妈，谢谢您，为一切的一切。感谢您为我和我哥哥所做的一切。感谢上帝，我有您这样的妈妈。

致我哥哥 Joe：每次我暂停写作从巴尔的摩（Baltimore）回家时，你总是提醒我：不工作的时候是多么美好，我应该早点完成写作，回到生活中更重要的事情去。你是个好人，我尊敬你。我特别为你感到自豪，也为能称你为哥哥而自豪。

致我的未婚妻 Georgia：如果没有你的支持，我不可能现在就完成了这本 600 多页的书。在本书的写作过程中，你一直陪伴着我，一天又一天。我知道，这期间你跟我一样艰辛。我白天在工作，晚上在写作，而你自始至终一直体贴我。你那么善解人意，我永远心存感激。谢谢你！我爱你！

致我未来的岳母 Kiki 和岳父 George：谢谢你们在这整个过程中对我的支持。每次休息一下去拜访你们时，我总能找到家的感觉，而且你们让我和 Georgia 衣食无忧。致妻妹 Anna

和Kathy：每次回家跟你们在一起我们总是很开心，这样使我和Georgia在工作和写作之余得到很好的放松。

致编辑Jonathan Gennick：没有你就没有这本书。Jonathan，这本书的大部分荣誉应属于你，你所做的远远超出了编辑的正常工作，为此，你应得到更多感谢。从提供素材到大量的推倒重来，尽管期限将至还是要保持内容的幽默，如果没有你，我不可能完成这一切。我很感谢你能做我的编辑，也很感谢你给我提供的机会。你自己就是DBA兼作者，能有机会跟有你这样技术水平和专长的人共事，我备感荣幸。不能想象有多少人能放弃专职编辑在其他地方做兼职DBA；但你就能。DBA的工作无疑给作为编辑的你带来优势，尽管我有时不知怎么表达，但你总是能知道我要说什么。有你这样的员工，O'Reilly很幸运；有你这样的编辑，我很幸运。

还要感谢Ales Spetic和Jonathan Gennick的Transact-SQL Cookbook一书。Isaac Newton有句名言：“如果说我看得比别人更远些，那是因为我站在巨人的肩膀上”。Ales Spetic在Transact-SQL Cookbook的“鸣谢”一节中写的一段文字是对这句名言的最好的注脚，我觉得每本SQL的书中都应该包含这一段，我也引用如下：

我希望这本书能对那些杰出作者的著作有所补充：如Joe Celko、David Rozenshtein、Anatoly Abramovich、Eugene Berger、Iztik Ben-Gan、Richard Snodgrass等。我成夜成夜地研读它们的著作，我所有的知识几乎都来自这些著作。我在写这些文字时我意识到：我每花一个晚上发现他们的秘密，他们必须花10个晚上将他们的知识表达成一致、易读的形式。能够为SQL世界回馈点什么是一种巨大的荣誉。

感谢Sanjay Mishra的卓越著作Mastering Oracle SQL，也感谢他让我认识了Jonathan。如果没有Sanjay，我可能就不会认识Jonathan，也就不可能写这本书。一份email就能改变一个人的生活，这多么令人惊讶！这里要特别感谢David Rozenshtein，他的书Essence of SQL让我对如何用集合/SQL的角度思考和解决问题打下了坚实的基础。还要感谢David Rozenshtein、Anatoly Abramovich和Eugene Birger的Optimizing Transact-SQL一书，我从中学到许多SQL高级技巧，至今仍在使用。

感谢Wireless Generation的团队，这是个好公司，有一帮好人。大力感谢花时间审阅、批评或提出建议以帮助我完成本书的所有人：esse Davis、Joel Patterson、Philip Zee、Kevin Marshall、Doug Daniels、Otis Gospodnetic、Ken Gunn、John Stewart、Jim Abramson、Adam Mayer、Susan Lau、Alexis Le-Quoc和Paul Feuer。感谢Maggie Ho，她审阅非常仔细，并对“窗口函数回顾”（附录A）提出了弥足珍贵的建议。感谢Chuck Van Buren和Gillian Gutenberg给我提出跑步的建议，清晨的外出活动帮助我清理思路，并使我得到放松，如果不是这种室外的调剂，这本书可能还没有完成。感谢Steve Kang，Chad Levinson，一整天工作之后他们只想到Union Square要杯啤酒，或到Heartland

Brewery 吃个汉堡，但是他们却花了许多晚上跟我不断讨论各种 SQL 技术。感谢 Aaron Boyd 给我的支持、安慰，更重要的是，良好的建议，Aaron 是个诚恳、不辞辛劳并且非常直爽的家伙，他这样的人会让一个公司更出色。还要感谢 Olivier Pomel 对写作本书的支持和帮助，尤其是“从行创建分隔列表”的 DB2 解决方案，Olivier 在没有 DB2 系统进行测试的情况下创建了这个解决方案，我只是跟他解释了一下 WITH 子句的工作机理，过了几分钟，他就拿出了本书中的解决方案。

Jonah Harris 和 David Rozenstein 也从技术的角度审阅了手稿，并提供了反馈，Arun Marathe 和 Nuno Pinto，在本书成型阶段，Andrew Odewahn 参与了大纲和方案选择方面的工作。在此一并感谢。

我要感谢 John Haydu 和 Oracle 公司的 MODEL 子句开发团队，他们花时间审阅了有关 MODEL 子句的部分，而且无疑使我加深了对该子句机理的理解，感谢 Oracle 公司的 Tom Kyte，他允许我将他的 TO\_BASE 函数改编成纯 SQL 的解决方案，Microsoft 的 Bruno Denuit 给我解答了有关 SQL Server 2005 中新引入的窗口函数的问题，PostgreSQL 的 Simon Riggs 让我了解了 PostgreSQL 的最新特性（非常感谢 Simon，知道了什么时候会有什么东西出来，我就可以将一些 SQL 的新特性结合近来，像非常棒的 GENERATE\_SERIES 函数，我认为用这个函数构成地解决方案比用基于表更雅致）。

最后，当然不是不重要，感谢 Key Young。你既有才能，又充满热情，能和你这样能干、热情的人共事真是很好。本书中许多方案是为解决 Wireless Generation 的日常问题跟 Key 一起得出的。我想让你知道，我非常感谢你给我的各方面支持：从建议到纠正语法错误，再到编码，你参与了这本书的整个创作过程。我很高兴能跟你一起工作，由于你的存在，Wireless Generation 变得更好了。

— Anthony Molinaro

# 检索记录

本章重点介绍基本的 SELECT 语句。本章内容非常重要，它不仅是许多复杂 SQL 语句的基础，而且是日常 SQL 工作中的常用内容，应当熟练掌握。

## 1.1 从表中检索所有行和列

### 问题

查看表中的所有数据。

### 解决方案

对表使用 SELECT 语句和特殊字符 “\*”：

```
1 select *  
2   from emp
```

### 讨论

在 SQL 中，字符 “\*” 具有特殊的含义。使用它，将从指定的表中返回每一列。这里由于没有使用 WHERE 子句，所以将会返回每一行。还有一种方法，就是分别列出每一列：

```
select empno,ename,job,sal,mgr,hiredate,comm,deptno  
from emp
```

在交互执行的特定查询中，使用 SELECT \* 更容易些。然而，在写程序代码的时候，最好是分别指定每一列。它们的性能相同，但是，显式地指定列后就可以很清楚查询中究竟返回了那些列。同样，其他人，而不是编写者自己，也更容易理解这样的查询（他们可能不知道查询中表的所有列）。

## 1.2 从表中检索部分行

### 问题

从表中查看满足特定条件的行。

## 解决方案

使用 WHERE 子句指定要保留哪些行。例如，要查看部门号码为 10 的所有员工：

```
1 select *
2   from emp
3  where deptno = 10
```

## 讨论

利用 WHERE 子句，可以只检索用户感兴趣的行。如果 WHERE 子句中的表达式对某行为真，则返回该行。

多数厂商都支持通用的运算符，例如 =、<、>、<=、>=、!=、<>。另外，如果要检索满足多种条件的行，可以使用 AND、OR 和圆括号，下一节将作介绍。

## 1.3 查找满足多个条件的行

### 问题

要返回满足多个条件的行。

### 解决方案

使用 WHERE 子句以及 OR 和 AND 子句。例如，如果要查找部门 10 中所有员工，所有得到提成的员工，以及部门 20 中工资不超过 2000 美金的员工，请使用如下语句：

```
1 select *
2   from emp
3  where deptno = 10
4         or comm is not null
5         or sal <= 2000 and deptno=20
```

### 讨论

可以使用 AND、OR 以及圆括号的组合，来查找满足多个条件的数据。在解决方案的示例中，WHERE 子句查找满足下列条件的数据行：

- DEPTNO 为 10, 或者
- COMM 不为 NULL, 或者
- 工资最多为 2000 美金, 且 DEPTNO 为 20

圆括号中的条件作为一个整体进行判断。

例如，考虑一下，用下面的方式使用圆括号，结果为什么会有变化：

```
select *
  from emp
 where (    deptno = 10
```

```

        or comm is not null
        or sal <= 2000
    )
    and deptno=20
EMPNO ENAME  JOB      MGR HIREDATE      SAL      COMM DEPTNO
-----
7369 SMITH  CLERK    7902 17-DEC-1980    800             20
7876 ADAMS  CLERK    7788 12-JAN-1983   1100             20

```

## 1.4 从表中检索部分列

### 问题

要查看一个表中特定列的值，而不是所有列的值。

### 解决方案

指定感兴趣的列。例如，如果只查看员工的名字、部门号和工资，请使用如下语句：

```

1 select ename,deptno,sal
2   from emp

```

### 讨论

通过在 SELECT 子句中指定列，可以保证不会返回多余的数据。在跨越网络检索数据时这非常重要，因为这样可以避免检索不需要的数据所带来的时间的浪费。

## 1.5 为列取有意义的名称

### 问题

改变查询所返回的列名，使它们更具可读性，更容易理解。。例如求每个员工的工资和提成的查询：

```

1 select sal,comm
2   from emp

```

什么是sal？是sale的缩写吗？还是某个人的名字？什么是comm？是表示 communication 吗？结果的标签应该更明确些。

### 解决方案

要改变查询结果列名，可以按这种格式使用 AS 关键字：原名 AS 新名。一些数据库不需要使用 AS，但所有的数据库都接受这种用法：

```

1 select sal as salary, comm as commission
2   from emp

```

SALARY	COMMISSION
800	
1600	300
1250	500
2975	

1250	1300
2850	
2450	
3000	
5000	
1500	0
1100	
950	
3000	
1300	

## 讨论

使用 AS 关键字可以为查询返回的列取个新名，即为这些列取别名，所取的新名就是别名。使用比较好的别名，可以使其他人更容易理解查询结果。

## 1.6 在 WHERE 子句中引用取别名的列

### 问题

前面已经使用别名功能，为查询结果提供了更有意义的列名，而且也使用 WHERE 子句将一些数据排除在外了，然而，我们还想在 WHERE 子句中引用别名：

```
select sal as salary, comm as commission
  from emp
 where salary < 5000
```

### 解决方案

将查询作为内联视图就可以引用其中区别名的列了：

```
1 select *
2   from (
3 select sal as salary, comm as commission
4   from emp
5   ) x
6  where salary < 5000
```

## 讨论

对这个简单的例子而言，可以不内联视图，也不在 WHERE 子句中直接引用 COMM 或 SAL 而得到相同的结果，本方案介绍的方法在下列情形的 WHERE 子句中都可以使用：

- 聚集函数
- 标量子查询
- 视窗函数
- 别名

将取别名的查询作为内联视图，便可以在外部查询中引用其中的别名列。为什么要这么做呢？WHERE 子句是在 SELECT 之前进行处理的，这样，在处理求解“问题”查询的 WHERE 子句之前，SALARY 和 COMMISSION 并不存在，要到 WHERE 子句处理完成之



后, 别名才生效。然而, FROM 子句是在 WHERE 之前处理的。将原查询放在 FROM 子句中, 那么, 在最外层的 WHERE 子句之前, 以及最外层的 WHERE 子句“看到”别名之前, 就已经生成了查询结果。如果表列没有特别命名的话, 这个技巧特别有用。

注意: 这个解决方案中的内联视图别名为 X。并非所有数据库都需要内联视图显式给内联视图取别名, 但一些数据库是这样的。所有的数据库都接受这种方式。

## 1.7 连接列值

### 问题

将多列值作为一列返回。例如, 要查询 EMP 表, 返回这样的结果集:

```
CLARK WORKS AS A MANAGER
KING WORKS AS A PRESIDENT
MILLER WORKS AS A CLERK
```

然而, 要得到这些数据, 它们来自两个不同的列, EMP 表中的 ENAME 和 JOB 列。

```
select ename, job
  from emp
 where deptno = 10

ENAME      JOB
-----
CLARK      MANAGER
KING       PRESIDENT
MILLER     CLERK
```

### 解决方案

查找和使用 DBMS 提供的内置函数, 来连接来自不同列的值。

#### DB2, Oracle, PostgreSQL

这些数据库使用双竖线作为连接运算符。

```
1 select ename||' WORKS AS A '||job as msg
2   from emp
3  where deptno=10
```

#### MySQL

这个数据库支持 CONCAT 函数。

```
1 select concat(ename, ' WORKS AS A ',job) as msg
2   from
3  where deptno=10
```

#### SQL Server

使用 “+” 运算符进行连接操作。

```
1 select ename + ' WORKS AS A ' + job as msg
2   from
3  where deptno=10
```

## 讨论

使用 CONCAT 函数连接来自多个列的数值。在 DB2、Oracle 和 PostgreSQL 中，“||”是 CONCAT 函数的简写方式，“+”是 SQL Server 中的简写方式。

## 1.8 在 SELECT 语句中使用条件逻辑

### 问题

要在 SELECT 语句中，对数值执行 IF-ELSE 操作。例如，要产生一个结果集，如果一个员工工资小于等于 2000 美金，就返回消息“UNDERPAID”；如果大于等于 4000 美金，就返回消息“OVERPAID”，如果是在这两者之间，就返回“OK”。结果集应如下所示：

ENAME	SAL	STATUS
SMITH	800	UNDERPAID
ALLEN	1600	UNDERPAID
WARD	1250	UNDERPAID
JONES	2975	OK
MARTIN	1250	UNDERPAID
BLAKE	2850	OK
CLARK	2450	OK
SCOTT	3000	OK
KING	5000	OVERPAID
TURNER	1500	UNDERPAID
ADAMS	1100	UNDERPAID
JAMES	950	UNDERPAID
FORD	3000	OK
MILLER	1300	UNDERPAID

### 解决方案

使用 CASE 表达式直接在 SELECT 语句中执行条件逻辑。

```
1 select ename,sal,
2        case when sal <= 2000 then 'UNDERPAID'
3             when sal >= 4000 then 'OVERPAID'
4             else 'OK'
5        end as status
6 from emp
```

## 讨论

CASE 表达式可以针对查询的返回值执行条件逻辑。可以给 CASE 表达式取别名，使结果集更易读。在这个解决方案中可以看到，给 CASE 表达式的别名是 STATUS。ELSE 子句是可选的，如果没有使用 ELSE，对于不满足判断条件的行，CASE 表达式会返回 NULL。

## 1.9 限制返回的行数

### 问题

限制查询中返回的行数。这里不关心顺序，返回任何  $n$  行都行。

## 解决方案

使用数据库提供的内置函数来控制返回的行数。

### DB2

在 DB2 中，使用 FETCH FIRST 子句：

```
1 select *
2   from emp fetch first 5 rows only
```

### MySQL 和 PostgreSQL

在 MySQL 和 PostgreSQL 中，使用 LIMIT：

```
1 select *
2   from emp limit 5
```

### Oracle

在 Oracle 中，在 WHERE 子句中通过使用 ROWNUM 来限制行数：

```
1 select *
2   from emp
3  where rownum <= 5
```

### SQL Server

使用 TOP 关键字，来限制返回的行数：

```
1 select top 5 *
2   from emp
```

## 讨论

许多数据库提供一些子句，比如 FETCH FIRST 和 LIMIT，让用户指定从查询中返回的行数。Oracle 的做法不同，必须使用 ROWNUM 函数来得到每行的行号（从 1 开始递增数值）。

在使用 ROWNUM <= 5 来返回前 5 行时，会发生下面的操作：

1. Oracle 执行查询。
2. Oracle 获取第 1 个符合条件的行，将它叫做第 1 行。
3. 有 5 行了吗？如果没有，那么，Oracle 就再返回行，因为它要满足行号小于等于 5 的条件，如果到了 5 行，那么，Oracle 就不再返回行。
4. Oracle 获取下一行，并递增行号（从 2，到 3，再到 4，等等）。
5. 返回到第 3 步。

可以看到，Oracle 的 ROWNUM 数值是在获取每行之后才赋予的。这非常重要，是一个

关键点。比如说，许多 Oracle 开发人员想通过指定 `ROWNUM = 5` 来返回第 5 行，这是错误的做法。下面说明使用 `ROWNUM = 5` 时会发生什么：

1. Oracle 执行查询。
2. Oracle 获取第 1 个符合条件的行，将它叫做第 1 行。
3. 有 5 行了吗？如果没有，那么，Oracle 就丢弃这些行，因为它不满足条件。如果到了 5 行，那么，Oracle 就返回该行。但是，答案是，永远也不会有“到了 5 行”的情况发生。
4. Oracle 获取下 1 行，这是第 1 行。原因是，从查询中返回的必须是编号为 1 的行。
5. 转向第 3 步。

仔细看看这个过程，可以知道使用 `ROWNUM = 5` 来返回第 5 行失败的原因。如果不返回第 1 行到第 4 行的话，就不会有第 5 行。

`ROWNUM = 1` 确实是返回第 1 行，这似乎与前面的说明矛盾了。原因是，`ROWNUM = 1` 返回第 1 行，不管表中是否有行，Oracle 都会尝试至少取 1 行。请仔细看前面叙述的过程，把 5 换为 1，就可以理解指定 `ROWNUM = 1` 作为条件来返回 1 行为什么是可行的了。

## 1.10 从表中随机返回 $n$ 条记录

### 问题

从表中随机返回  $n$  条记录。可以修改下面的语句，要求下次执行时产生不同的结果集。

```
select ename, job
from emp
```

### 解决方案

使用 DBMS 支持的内置函数来生成随机数值。在 `ORDER BY` 子句中使用该函数，对行进行随机排序，然后，使用前面问题介绍的技巧，来限制所返回的行（顺序随机）的数目。

#### DB2

同时使用内置函数 `RAND` 与 `ORDER BY` 和 `FETCH`。

```
1 select ename, job
2   from emp
3  order by rand() fetch first 5 rows only
```

#### MySQL

同时使用内置的 `RAND` 函数、`LIMIT` 和 `ORDER BY`：

```
1 select ename, job
2   from emp
3  order by rand() limit 5
```

## PostgreSQL

同时使用内置 RANDOM 函数、LIMIT 和 ORDER BY:

```
1 select ename, job
2   from emp
3  order by random() limit 5
```

## Oracle

同时使用 DBMS\_RANDOM 包中的内置函数 VALUE、ORDER BY 和内置函数 ROWNUM:

```
1 select *
2   from (
3     select ename, job
4       from emp
5     order by dbms_random.value()
6   )
7  where rownum <= 5
```

## SQL Server

同时使用内置函数 NEWID、TOP 和 ORDER BY, 返回随机结果集:

```
1 select top 5 ename, job
2   from emp
3  order by newid()
```

## 讨论

ORDER BY 子句可以接受函数的返回值, 并使用它来改变结果集的次序。这个解决方案中, 在 ORDER BY 子句中执行函数之后, 再查询返回的行数。非 Oracle 用户会发现, 看看 Oracle 解决方案会很有用, 可以理解解决方案的原理。

重要的是, 不要把 ORDER BY 子句中使用函数与使用数字常量混淆起来。在 ORDER BY 子句中指定数字常量时, 是要求根据 SELECT 列表中相应位置的列来排序, 在 ORDER BY 子句中使用函数时, 则按函数在每一行计算结果排序。

### 1.11 查找空值

#### 问题

要查找某列值为空的所有行。

#### 解决方案

要确定值是否为空, 必须使用 IS NULL:

```
1 select *
2   from emp
3  where comm is null
```

## 讨论

NULL 不能用等于或不等于跟任何值比较，包括它自身。所以，不能使用 = 或 != 来测试一列是否为 NULL。为了确定一行是否有空值，必须使用 IS NULL，也可以使用 IS NOT NULL 来查找给定列的值不为空的行。

## 1.12 将空值转换为实际值

### 问题

在一些行中包含空值，需要使用非空值来代替这些空值。

### 解决方案

使用 COALESCE 函数用实际的值来替换空值，语句如下：

```
1 select coalesce(comm,0)
2   from emp
```

## 讨论

COALESCE 函数有 1 个或多个参数。该函数返回列表中的第一个非空值。在这个解决方案中，只要 COMM 非空，就返回 COMM 的值，否则返回 0。

在使用空值的时候，最好是利用 DBMS 提供的内置功能，许多情况下，有几个函数都可以完成这项任务。COALESCE 可以用于所有的 DBMS。另外，对于所有的 DBMS，也都可以使用 CASE，如下所示：

```
select case
      when comm is null then 0
      else comm
    end
  from emp
```

尽管可以使用 CASE 将空值转换为非空数值，但是，可以看到，使用 COALESCE 更为容易、简洁。

## 1.13 按模式搜索

### 问题

需要返回匹配特定子串或模式的行。考虑下面的查询和结果集：

```
select ename, job
  from emp
 where deptno in (10,20)
```

ENAME	JOB
SMITH	CLERK
JONES	MANAGER
CLARK	MANAGER
SCOTT	ANALYST
KING	PRESIDENT
ADAMS	CLERK
FORD	ANALYST
MILLER	CLERK

在部门 10 和部门 20，需要返回名字中有一个 “I” 或者职务 (job title) 中带有 “ER” 的员工。

ENAME	JOB
SMITH	CLERK
JONES	MANAGER
CLARK	MANAGER
KING	PRESIDENT
MILLER	CLERK

## 解决方案

使用 LIKE 运算符和 SQL 通配符 “%”。

```
1 select ename, job
2   from emp
3  where deptno in (10,20)
4     and (ename like '%I%' or job like '%ER')
```

## 讨论

在 LIKE 模式匹配操作中，百分号 “%” 运算符可以匹配任何字符序列。多数 SQL 实现中也提供了下划线 “\_” 运算符，来匹配单个字符。使用 “%” 运算符将搜索模式 “I” 括起来，就会返回任何包含 “I” 的字符串，不管 “I” 在什么位置。如果不用 “%” 运算符将搜索模式 “I” 括起来，那么，这个运算符的位置就会影响查询结果。例如，要查找以 “ER” 结尾的职务，可以在 “ER” 的前面加上前缀 “%” 运算符；如果需要查找以 “ER” 起始的职务，则将 “%” 放在 “ER” 的后面。



## 第2章

# 查询结果排序

本章重点介绍如果自定义查询结果的外观。理解了如何控制和修改结果集，便可以提供可读性更好而且更有意义的数据库。

### 2.1 以指定的次序返回查询结果

#### 问题

显示部门 10 中的员工名字、职位和工资，并按照工资的升序排列。结果集如下：

ENAME	JOB	SAL
MILLER	CLERK	1300
CLARK	MANAGER	2450
KING	PRESIDENT	5000

#### 解决方案

使用 ORDER BY 子句：

```
1 select ename,job,sal
2   from emp
3  where deptno = 10
4  order by sal asc
```

#### 讨论

使用 ORDER BY 子句可以对结果集进行排序。解决方案按 SAL 的升序对行进行排列。默认情况下，ORDER BY 以升序方式排序，因此 ASC 子句是可选的。DESC 表示降序排列：

```
select ename,job,sal
  from emp
 where deptno = 10
 order by sal desc
```

ENAME	JOB	SAL
KING	PRESIDENT	5000
CLARK	MANAGER	2450
MILLER	CLERK	1300

不一定要指定排序所基于的列名，也可以给出表示这列的编号。这个编号从 1 开始，从左到右依次对应 SELECT 列表中的各项目。例如：

```
select ename,job,sal
  from emp
 where deptno = 10
 order by 3 desc
```

ENAME	JOB	SAL
KING	PRESIDENT	5000
CLARK	MANAGER	2450
MILLER	CLERK	1300

例中 ORDER BY 子句中的 3，与 SELECT 列表中的第 3 列对应，也就是 SAL。

## 2.2 按多个字段排序

### 问题

在 EMP 表中，首先按照 DEPTNO 的升序排序行，然后按照工资的降序排列，结果集应如下：

EMPNO	DEPTNO	SAL	ENAME	JOB
7839	10	5000	KING	PRESIDENT
7782	10	2450	CLARK	MANAGER
7934	10	1300	MILLER	CLERK
7788	20	3000	SCOTT	ANALYST
7902	20	3000	FORD	ANALYST
7566	20	2975	JONES	MANAGER
7876	20	1100	ADAMS	CLERK
7369	20	800	SMITH	CLERK
7698	30	2850	BLAKE	MANAGER
7499	30	1600	ALLEN	SALESMAN
7844	30	1500	TURNER	SALESMAN
7521	30	1250	WARD	SALESMAN
7654	30	1250	MARTIN	SALESMAN
7900	30	950	JAMES	CLERK

### 解决方案

在 ORDER BY 子句中列出不同的排序列，使用逗号分隔：

```
1 select empno,deptno,sal,ename,job
2   from emp
3  order by deptno, sal desc
```

### 讨论

在 ORDER BY 中，优先次序是从左到右。如果在 SELECT 列表中使用列的数字位置排序，那么，这个数值不能大于 SELECT 列表中项目的数目。一般情况下都可以按照 SELECT 列表中没有的列来排序，但是必须显式地给出排序的列名。而如果在查询中使用 GROUP BY 或 DISTINCT，则不能按照 SELECT 列表中没有的列来排序。

## 2.3 按子串排序

### 问题

按字符串的某一部分对查询结果排序。例如，要从EMP表中返回员工名字和职位，并且按照职位字段的最后两个字符排序，结果集应如下：

ENAME	JOB
KING	PRESIDENT
SMITH	CLERK
ADAMS	CLERK
JAMES	CLERK
MILLER	CLERK
JONES	MANAGER
CLARK	MANAGER
BLAKE	MANAGER
ALLEN	SALESMAN
MARTIN	SALESMAN
WARD	SALESMAN
TURNER	SALESMAN
SCOTT	ANALYST
FORD	ANALYST

### 解决方案

#### DB2、MySQL、Oracle 和 PostgreSQL

在 ORDER BY 子句中使用 SUBSTR 函数：

```
select ename,job
  from emp
 order by substr(job,length(job)-2)
```

#### SQL Server

在 ORDER BY 子句中使用 substring 函数：

```
select ename,job
  from emp
 order by substring(job,len(job)-2,2)
```

### 讨论

使用DBMS的子串函数，可以很容易地按字符串的一部分来排序。要按照字符串的最后两个字符排序，首先要找到字符串的末尾（就是字符串的长度），并减去2。起始位置就是字符串中的倒数第2个字符。然后，获取从起始位置开始的所有字符。SQL Server在SUBSTRING中需要第3个参数来指定要获取的字符数。在本例中，只要这个数目大于或等于2就可以。

## 2.4 对字母数字混合的数据排序

### 问题

现有字母和数字混合的数据，希望按照数字或字符部分来排序。考虑这个视图：

```
create view v
as
select ename||' '||deptno as data
from emp

select * from v

DATA
-----
SMITH 20
ALLEN 30
WARD 30
JONES 20
MARTIN 30
BLAKE 30
CLARK 10
SCOTT 20
KING 10
TURNER 30
ADAMS 20
JAMES 30
FORD 20
MILLER 10
```

要通过 DEPTNO 或 ENAME 来排序结果。通过 DEPTNO 排序可以产生下面的结果集：

```
DATA
-----
CLARK 10
KING 10
MILLER 10
SMITH 20
ADAMS 20
FORD 20
SCOTT 20
JONES 20
ALLEN 30
BLAKE 30
MARTIN 30
JAMES 30
TURNER 30
WARD 30
```

通过 ENAME 排序，产生下面的结果集：

```
DATA
-----
ADAMS 20
ALLEN 30
BLAKE 30
CLARK 10
FORD 20
JAMES 30
JONES 20
KING 10
MARTIN 30
MILLER 10
SCOTT 20
SMITH 20
TURNER 30
WARD 30
```

## 解决方案

### Oracle 和 PostgreSQL

使用函数 REPLACE 和 TRANSLATE 修改要排序的字符串：  
[www.TopSage.com](http://www.TopSage.com)

```

/* ORDER BY DEPTNO */
1 select data
2   from V
3   order by replace(data,
4     replace(
5       translate(data,'0123456789','#####'),'#',''), '')
/* ORDER BY ENAME */
1 select data
2   from emp
3   order by replace(
4     translate(data,'0123456789','#####'),'#','')

```

## DB2

在 DB2 中，隐式类型转换比在 Oracle 或 PostgreSQL 中更为严格。所以，为了使视图 V 有效，需要将 DEPTNO 转换为 CHAR 类型。本解决方案不重新创建视图 V，只是使用内联视图。本解决方案跟 Oracle 和 PostgreSQL 方案中使用 REPLACE 和 TRANSLATE 的方式基本相同，只是 TRANSLATE 参数的次序不同：

```

/* ORDER BY DEPTNO */
1 select *
2   from (
3 select ename||' '||cast(deptno as char(2)) as data
4   from emp
5   ) v
6   order by replace(data,
7     replace(
8       translate(data,'#####','0123456789'),'#',''), '')
/* ORDER BY ENAME */
1 select *
2   from (
3 select ename||' '||cast(deptno as char(2)) as data
4   from emp
5   ) v
6   order by replace(
7     translate(data,'#####','0123456789'),'#','')

```

## MySQL 和 SQL Server

这些平台当前不支持 TRANSLATE 函数，这个问题无解决方案。

## 讨论

TRANSLATE 和 REPLACE 函数从每一行中去掉数字或字符，这样就可以很容易地根据具体情况来排序。传递给 ORDER BY 的值在随后的查询结果中显示出来（使用 Oracle 解决方案为例，三个平台都使用相同的技术。只有传递给 TRANSLATE 的参数顺序与 DB2 不同）：

```

select data,
       replace(data,
               replace(
                 translate(data,'0123456789','#####'),'#',''), '') nums,
       replace(
         translate(data,'0123456789','#####'),'#','') chars
from V

```

DATA

NUMS

CHARS

www.TopSage.com

SMITH 20	20	SMITH
ALLEN 30	30	ALLEN
WARD 30	30	WARD
JONES 20	20	JONES
MARTIN 30	30	MARTIN
BLAKE 30	30	BLAKE
CLARK 10	10	CLARK
SCOTT 20	20	SCOTT
KING 10	10	KING
TURNER 30	30	TURNER
ADAMS 20	20	ADAMS
JAMES 30	30	JAMES
FORD 20	20	FORD
MILLER 10	10	MILLER

## 2.5 处理排序空值

### 问题

在 EMP 中根据 COMM 排序结果。但是，这个字段可以有空值。需要指定是否将空值排在最后。

ENAME	SAL	COMM
TURNER	1500	0
ALLEN	1600	300
WARD	1250	500
MARTIN	1250	1400
SMITH	800	
JONES	2975	
JAMES	950	
MILLER	1300	
FORD	3000	
ADAMS	1100	
BLAKE	2850	
CLARK	2450	
SCOTT	3000	
KING	5000	

或者，是否将空值排在最前面。

ENAME	SAL	COMM
SMITH	800	
JONES	2975	
CLARK	2450	
BLAKE	2850	
SCOTT	3000	
KING	5000	
JAMES	950	
MILLER	1300	
FORD	3000	
ADAMS	1100	
MARTIN	1250	1400
WARD	1250	500
ALLEN	1600	300
TURNER	1500	0

### 解决方案

根据数据的显示方式，以及特定的 RDBMS 排序空值的方式，可以按照升序或降序来对有空值的列排序。

```

1 select ename,sal,comm
2   from emp
3  order by 3

```

```

1 select ename,sal,comm
2   from emp
3  order by 3 desc

```

这种解决方案中，如果可为空值的列包含非空值，那么也可以根据要求，按升序或降序排序，这可能正是所期待的，也许不是。如果希望空值的排序与非空值不同，例如，要以升序或降序方式来排序非空值，将空值放在最后，则可使用CASE表达式有条件地排序列。

## DB2、MySQL、PostgreSQL 和 SQL Server

使用CASE表达式来“标记”一个值是否为NULL。这个标记有两个值，一个表示NULL，一个表示非NULL。这样，只要在ORDER BY子句中增加标记列，便可以很容易地控制空值是排在前面还是排在最后，而不会被非空值所干扰。

```
/* NON-NULL COMM SORTED ASCENDING, ALL NULLS LAST */
```

```

1 select ename,sal,comm
2   from (
3 select ename,sal,comm,
4        case when comm is null then 0 else 1 end as is_null
5   from emp
6  ) x
7  order by is_null desc,comm

```

ENAME	SAL	COMM
TURNER	1500	0
ALLEN	1600	300
WARD	1250	500
MARTIN	1250	1400
SMITH	800	
JONES	2975	
JAMES	950	
MILLER	1300	
FORD	3000	
ADAMS	1100	
BLAKE	2850	
CLARK	2450	
SCOTT	3000	
KING	5000	

```
/* NON-NULL COMM SORTED DESCENDING, ALL NULLS LAST */
```

```

1 select ename,sal,comm
2   from (
3 select ename,sal,comm,
4        case when comm is null then 0 else 1 end as is_null
5   from emp
6  ) x
7  order by is_null desc,comm desc

```

ENAME	SAL	COMM
MARTIN	1250	1400
WARD	1250	500
ALLEN	1600	300
TURNER	1500	0
SMITH	800	



```

JONES    2975
JAMES     950
MILLER   1300
FORD     3000
ADAMS    1100
BLAKE    2850
CLARK    2450
SCOTT    3000
KING     5000

```

```
/* NON-NULl COMM SORTED ASCENDING, ALL NULLS FIRST */
```

```

1 select ename,sal,comm
2   from (
3 select ename,sal,comm,
4        case when comm is null then 0 else 1 end as is_null
5   from emp
6   ) x
7  order by is_null,comm

```

ENAME	SAL	COMM
SMITH	800	
JONES	2975	
CLARK	2450	
BLAKE	2850	
SCOTT	3000	
KING	5000	
JAMES	950	
MILLER	1300	
FORD	3000	
ADAMS	1100	
TURNER	1500	0
ALLEN	1600	300
WARD	1250	500
MARTIN	1250	1400

```
/* NON-NULl COMM SORTED DESCENDING, ALL NULLS FIRST */
```

```

1 select ename,sal,comm
2   from (
3 select ename,sal,comm,
4        case when comm is null then 0 else 1 end as is_null
5   from emp
6   ) x
7  order by is_null,comm desc

```

ENAME	SAL	COMM
SMITH	800	
JONES	2975	
CLARK	2450	
BLAKE	2850	
SCOTT	3000	
KING	5000	
JAMES	950	
MILLER	1300	
FORD	3000	
ADAMS	1100	
MARTIN	1250	1400
WARD	1250	500
ALLEN	1600	300
TURNER	1500	0

## Oracle

Oracle8i Database 以及较早版本的用户可以使用跟其他平台相同的解决方案。Oracle9i  
[www.TopSage.com](http://www.TopSage.com)

Database 及以后版本的用户可以在 ORDER BY 子句中使用 NULLS FIRST 或 NULLS LAST，来确保 NULL 是首先排序，还是最后排序，而不必考虑非空值的排序方式。

```
/* NON-NULL COMM SORTED ASCENDING, ALL NULLS LAST */
```

```
1 select ename,sal,comm
2   from emp
3  order by comm nulls last
```

ENAME	SAL	COMM
TURNER	1500	0
ALLEN	1600	300
WARD	1250	500
MARTIN	1250	1400
SMITH	800	
JONES	2975	
JAMES	950	
MILLER	1300	
FORD	3000	
ADAMS	1100	
BLAKE	2850	
CLARK	2450	
SCOTT	3000	
KING	5000	

```
/* NON-NULL COMM SORTED DESCENDING, ALL NULLS LAST */
```

```
1 select ename,sal,comm
2   from emp
3  order by comm desc nulls last
```

ENAME	SAL	COMM
MARTIN	1250	1400
WARD	1250	500
ALLEN	1600	300
TURNER	1500	0
SMITH	800	
JONES	2975	
JAMES	950	
MILLER	1300	
FORD	3000	
ADAMS	1100	
BLAKE	2850	
CLARK	2450	
SCOTT	3000	
KING	5000	

```
/* NON-NULL COMM SORTED ASCENDING, ALL NULLS FIRST */
```

```
1 select ename,sal,comm
2   from emp
3  order by comm nulls first
```

ENAME	SAL	COMM
SMITH	800	
JONES	2975	
CLARK	2450	
BLAKE	2850	
SCOTT	3000	
KING	5000	
JAMES	950	
MILLER	1300	
FORD	3000	
ADAMS	1100	
TURNER	1500	

ALLEN	1600	300
WARD	1250	500
MARTIN	1250	1400

```
/* NON-NULL COMM SORTED DESCENDING, ALL NULLS FIRST */
```

```
1 select ename,sal,comm
2   from emp
3  order by comm desc nulls first
```

ENAME	SAL	COMM
SMITH	800	
JONES	2975	
CLARK	2450	
BLAKE	2850	
SCOTT	3000	
KING	5000	
JAMES	950	
MILLER	1300	
FORD	3000	
ADAMS	1100	
MARTIN	1250	1400
WARD	1250	500
ALLEN	1600	300
TURNER	1500	0

## 讨论

除非 RDBMS 提供了一种方式，可以很容易地将空值排在最前或排在最后，而不必在同一列修改非空值（例如 Oracle 的情况），否则，就需要一个附加的列。

---

**注意：**到编写本书的时候，DB2 用户可以在窗口函数的 OVER 字句的 ORDER BY 子句中使用 NULLS FIRST 和 NULLS LAST，而不是在 ORDER BY 子句中使用以作用于整个结果集。

---

多加 1 列的目的是（只是在查询中，不是在表中），可以判别空值，并将它们排在一起，放在最前面或最后面。对于非 Oracle 解决方案，下面的查询返回内联视图 X 的结果集：

```
select ename,sal,comm,
       case when comm is null then 0 else 1 end as is_null
  from emp
```

ENAME	SAL	COMM	IS_NULL
SMITH	800		0
ALLEN	1600	300	1
WARD	1250	500	1
JONES	2975		0
MARTIN	1250	1400	1
BLAKE	2850		0
CLARK	2450		0
SCOTT	3000		0
KING	5000		0
TURNER	1500	0	1
ADAMS	1100		0
JAMES	950		0
FORD	3000		0
MILLER	1300		0

根据 IS\_NULL 的值, 可以很容易地将 NULL 排在最前面或最后面, 而不影响 COMM 如何排序。

## 2.6 根据数据项的键排序

### 问题

要根据某些条件逻辑来排序。例如, 如果 JOB 是 "SALESMAN", 要根据 COMM 来排序。否则, 根据 SAL 排序。要返回下面的结果集:

ENAME	SAL	JOB	COMM
TURNER	1500	SALESMAN	0
ALLEN	1600	SALESMAN	300
WARD	1250	SALESMAN	500
SMITH	800	CLERK	
JAMES	950	CLERK	
ADAMS	1100	CLERK	
MARTIN	1250	SALESMAN	1300
MILLER	1300	CLERK	
CLARK	2450	MANAGER	
BLAKE	2850	MANAGER	
JONES	2975	MANAGER	
SCOTT	3000	ANALYST	
FORD	3000	ANALYST	
KING	5000	PRESIDENT	

### 解决方案

在 ORDER BY 子句中使用 CASE 表达式:

```
1 select ename,sal,job,comm
2   from emp
3  order by case when job = 'SALESMAN' then comm else sal end
```

### 讨论

可以使用 CASE 表达式来动态改变如何对结果排序。传递给 ORDER BY 的值类似这样:

```
select ename,sal,job,comm,
       case when job = 'SALESMAN' then comm else sal end as ordered
  from emp
 order by 5
```

ENAME	SAL	JOB	COMM	ORDERED
TURNER	1500	SALESMAN	0	0
ALLEN	1600	SALESMAN	300	300
WARD	1250	SALESMAN	500	500
SMITH	800	CLERK		800
JAMES	950	CLERK		950
ADAMS	1100	CLERK		1100
MARTIN	1250	SALESMAN	1300	1300
MILLER	1300	CLERK		1300
CLARK	2450	MANAGER		2450
BLAKE	2850	MANAGER		2850
JONES	2975	MANAGER		2975
SCOTT	3000	ANALYST		3000
FORD	3000	ANALYST		3000
KING	5000	PRESIDENT		5000

## 操作多个表

本章介绍如何使用联接和集合操作，将多个表的数据组合在一起。联接是 SQL 的基础。集合操作也非常重要。本书后续章节的复杂查询，都以本章的联接和集合操作为基础。

### 3.1 记录集的叠加

#### 问题

要将来自多个表的数据组织到一起，就像将一个结果集叠加到另一个上面一样。这些表不必有相同的关键字，但是，它们对应列的数据类型应相同。例如，要显示 EMP 表部门 10 中员工的名字和部门编号，以及 DEPT 表中每个员工的名字和部门编号，结果集如下所示。

ENAME_AND_DNAME	DEPTNO
-----	-----
CLARK	10
KING	10
MILLER	10
-----	
ACCOUNTING	10
RESEARCH	20
SALES	30
OPERATIONS	40

#### 解决方案

使用集合操作 UNION ALL 把多个表中的行组合到一起。

```

1  select ename as ename_and_dname, deptno
2  from emp
3  where deptno = 10
4  union all
5  select '-----', null
6  from t1
7  union all
8  select dname, deptno
9  from dept

```

## 讨论

UNION ALL 将多个来源的行组合起来，放到一个结果集中。所有 SELECT 列表中的项目数和对应项目的数据类型必须要匹配，这跟其他所有集合的操作要求相同。例如，下面的两个查询将会失败：

```

select deptno | select deptno, dname
  from dept   |   from dept
union all     |   union
select ename  | select deptno
  from emp    |   from emp

```

务请注意，UNION ALL 将包括重复的项目。如果要筛选掉重复项，可以使用 UNION 运算符。例如，EMP.DEPTNO 和 DEPT.DEPTNO 之间的 UNION 将只返回 4 行。

```

select deptno
  from emp
union
select deptno
  from dept

DEPTNO
-----
      10
      20
      30
      40

```

如果使用 UNION 而不是 UNION ALL，很可能会为了去除重复项而进行排序操作。在处理大结果集时要记住，使用 UNION 子句大致等价于下面的查询，对 UNION ALL 子句的查询结果使用 DISTINCT：

```

select distinct deptno
  from (
select deptno
  from emp
union all
select deptno
  from dept
)

DEPTNO
-----
      10
      20
      30
      40

```

通常，查询中不要使用 DISTINCT，除非确有必要这样做；对于 UNION 而言也是如此，除非确有必要，一般使用 UNION ALL，而不使用 UNION。

## 3.2 组合相关的行

### 问题

多个表有一些相同列，或有些列的值相同，要通过联接这些列得到结果。例如，要显示部门 10 中所有员工的名字，以及每个员工所在部门的工作地点，这些数据存储在两个独立的表中。其结果集应如下所示。

ENAME	LOC
CLARK	NEW YORK
KING	NEW YORK
MILLER	NEW YORK

## 解决方案

根据 DEPTNO 联接表 EMP 和 DEPT:

```
1 select e.ename, d.loc
2   from emp e, dept d
3  where e.deptno = d.deptno
4     and e.deptno = 10
```

## 讨论

该解决方案是联接的一种,更准确地说是等值联接 (equi-join),这是内联接 (inner join) 的一种类型。联接操作会将来自两个表的行组合到一个表中。等值联接的联接条件是相等条件,比如说,两个表的部门编号相等。内联接是联接的原始类型,返回的每一行都包含来自每个表的数据。

从概念上来说,要得到联接的结果集,首先要创建 FORM 子句后列出的表的笛卡儿积 (行的所有可能组合),如下所示:

```
select e.ename, d.loc,
       e.deptno as emp_deptno,
       d.deptno as dept_deptno
  from emp e, dept d
 where e.deptno = 10
```

ENAME	LOC	EMP_DEPTNO	DEPT_DEPTNO
CLARK	NEW YORK	10	10
KING	NEW YORK	10	10
MILLER	NEW YORK	10	10
CLARK	DALLAS	10	20
KING	DALLAS	10	20
MILLER	DALLAS	10	20
CLARK	CHICAGO	10	30
KING	CHICAGO	10	30
MILLER	CHICAGO	10	30
CLARK	BOSTON	10	40
KING	BOSTON	10	40
MILLER	BOSTON	10	40

这将返回表 EMP 部门 10 中的所有员工,以及表 DEPT 的所有部门。然后,在 WHERE 子句的表达式中使用 e.deptno 和 d.deptno 来限制结果集,只返回 EMP.DEPTNO 和 DEPT.DEPTNO 相等的那些行。

```
select e.ename, d.loc,
       e.deptno as emp_deptno,
       d.deptno as dept_deptno
  from emp e, dept d
 where e.deptno = d.deptno
       and e.deptno = 10
```



ENAME	LOC	EMP_DEPTNO	DEPT_DEPTNO
CLARK	NEW YORK	10	10
KING	NEW YORK	10	10
MILLER	NEW YORK	10	10

还有一种解决方案，就是利用显式的 JOIN 子句（INNER 关键字可选）：

```
select e.ename, d.loc
  from emp e inner join dept d
    on (e.deptno = d.deptno)
 where e.deptno = 10
```

如果希望将联接逻辑放在 FROM 子句中，而不是在 WHERE 子句中，可以使用 JOIN 子句。这两种方式都符合 ANSI 标准，而且在本书涉及的所有 RDBMS 的最新版本中都适用。

### 3.3 在两个表中查找共同行问题

查找两个表中共同行，但有多列可以用来联接的这两个表。例如，考虑下面的视图 V：

```
create view V
as
select ename, job, sal
  from emp
 where job = 'CLERK'
select * from V
```

ENAME	JOB	SAL
SMITH	CLERK	800
ADAMS	CLERK	1100
JAMES	CLERK	950
MILLER	CLERK	1300

视图 V 只返回了职员的数据，然而，这个视图并没有显示出 EMP 的所有列。现需要返回表 EMP 中与视图得到的行相匹配的所有职员的 EMPNO、ENAME、JOB、SAL 和 DEPTNO，得到如下结果集：

EMPNO	ENAME	JOB	SAL	DEPTNO
7369	SMITH	CLERK	800	20
7876	ADAMS	CLERK	1100	20
7900	JAMES	CLERK	950	30
7934	MILLER	CLERK	1300	10

#### 解决方案

要返回正确的结果，必须按所有必要的列进行联接。或者，如果不想进行联接，也可以使用集合操作 INTERSECT，返回两个表的交集（共同的行）。

#### MySQL 和 SQL Server

使用多个联接条件，将表 EMP 与视图 V 联接起来。

```
1 select e.empno, e.ename, e.job, e.sal, e.deptno
```

```
2   from emp e, V
3   where e.ename = v.ename
4         and e.job  = v.job
5         and e.sal   = v.sal
```

另外，还可以通过 JOIN 子句执行相同的联接。

```
1  select e.empno,e.ename,e.job,e.sal,e.deptno
2    from emp e join V
3        on (   e.ename = v.ename
4              and e.job  = v.job
5              and e.sal   = v.sal )
```

## DB2、Oracle 和 PostgreSQL

MySQL 和 SQL Server 解决方案也可以用于 DB2、Oracle 和 PostgreSQL。如果需要从视图 V 中返回值，就要使用这个解决方案。

如果实际上并不需要从视图 V 中返回列，可以使用集合操作 INTERSECT 以及 IN 谓词：

```
1  select empno,ename,job,sal,deptno
2    from emp
3   where (ename,job,sal) in (
4     select ename,job,sal from emp
5    intersect
6     select ename,job,sal from V
7   )
```

## 讨论

在执行联接时，为了返回正确的结果，必须考虑按适当的列联接。当某些列中含有相同的值，而其他列中含有不同的值时，这一点尤其重要。

集合操作 INTERSECT 将返回两个行来源中的共同行。在使用 INTERSECT 时，要求两个表的项目数目相同，对应的数据类型也相同。使用集合操作的时候要注意，默认情况下，不会返回重复行。

## 3.4 从一个表中查找另一个表没有的值问题

要从一个表（称之为源表）中查找在另一目标表中不存在的值。例如，要从表 DEPT 中查找在表 EMP 中不存在数据的所有部门。在示例数据中，DEPTNO 值为 40 的记录在表 EMP 中不存在，所以结果集应该如下所示：

```
DEPTNO
-----
40
```

## 解决方案

求差集函数对解决这个问题非常有用。DB2、PostgreSQL 和 Oracle 支持差集操作。如果 DBMS 不支持差集函数，可以像 MySQL 和 SQL Server 方案一样使用子查询。

## DB2 和 PostgreSQL

使用集合操作 EXCEPT:

```
1 select deptno from dept
2 except
3 select deptno from emp
```

## Oracle

使用集合操作 MINUS:

```
1 select deptno from dept
2 minus
3 select deptno from emp
```

## MySQL 和 SQL Server

使用子查询返回表 EMP 中所有的 DEPTNO, 而外层查询则从 DEPT 表中查找子查询的结果中所没有的行:

```
1 select deptno
2   from dept
3  where deptno not in (select deptno from emp)
```

## 讨论

### DB2 和 PostgreSQL

用 DB2 和 PostgreSQL 提供的内置函数可以很容易地完成这种操作。EXCEPT 操作符获取第一个结果集, 并从中去掉那些第二个结果集中也有的行。这种操作与减法十分相似。

EXCEPT 的使用也有一定的限制, 两个 SELECT 列表中值的数目和数据类型必须匹配, 此外, EXCEPT 不会返回重复行, 而且跟使用 NOT IN 的子查询不同, NULL 值不会有问题 (参见有关 MySQL 和 SQL Server 的讨论)。EXCEPT 操作符将从前一个查询 (EXCEPT 之前的查询) 的结果中返回所有在后一个查询 (EXCEPT 之后的查询) 结果中所没有的行。

## Oracle

Oracle 与 DB2 和 PostgreSQL 的解决方案一样, 只是在 Oracle 中差集的操作符为 MINUS, 而不是 EXCEPT, 除此之外, 前面的解释也同样适用于 Oracle。

## MySQL 和 SQL Server

在子查询中, 将返回表 EMP 中的所有 DEPTNO 值, 而外层查询将返回 DEPT 表中满足如下条件的 DEPTNO 值: “不在” 或 “不包含在” 子查询结果集中。

当使用 MySQL 和 SQL Server 解决方案时, 应当考虑如何删除重复行。其他平台基于 EXCEPT 和 MINUS 的解决方案从结果集中去除了重复行, 从而确保每个 DEPTNO 值在结果中只出现一次。当然, 本例是不会有重复值的, 因为在示例数据中 DEPTNO 是关键

字字段。如果 DEPTNO 不是关键字，可以使用 DISTINCT 子句来确保每个在 EMP 表中没有的 DEPTNO 值只在结果中出现一次。

```
select distinct deptno
  from dept
 where deptno not in (select deptno from emp)
```

当使用 NOT IN 子句时，一定要注意 Null 值的问题。考虑下面的表 NEW\_DEPT：

```
create table new_dept(deptno integer)
insert into new_dept values (10)
insert into new_dept values (50)
insert into new_dept values (null)
```

如果使用子查询和 NOT IN 子句，来查找表 DEPT 中所有在 NEW\_DEPT 表中没有的 DEPTNO 值，将会发现返回的查询结果是空的。

```
select *
  from dept
 where deptno not in (select deptno from new_dept)
```

DEPTNO 值为 20、30 和 40 的记录在表 NEW\_DEPT 并不存在，但这个查询没有返回结果。原因就是在表 NEW\_DEPT 中有一个空值。子查询返回的 3 行结果集中，DEPTNO 的值分别为 10、50 和 NULL。IN 和 NOT IN 本质上是 OR 运算，因而计算逻辑 OR 时处理 NULL 的方式不同，产生的结果也不同。考虑下面列出的几个例子，其中分别使用了 IN 和与其等价的 OR 操作符：

```
select deptno
  from dept
 where deptno in ( 10,50,null )

DEPTNO
-----
    10

select deptno
  from dept
 where ( deptno=10 or deptno=50 or deptno=null)

DEPTNO
-----
    10
```

现在考虑同样的例子，分别使用了 NOT IN 和 NOT OR 操作符：

```
select deptno
  from dept
 where deptno not in ( 10,50,null )

( 空行)

select deptno
  from dept
 where not (deptno=10 or deptno=50 or deptno=null)

( 空行)
```

可以看到，条件 DEPTNO NOT IN (10, 50, NULL) 等同于：

```
not (deptno=10 or deptno=50 or deptno=null)
```

在这种情况下，当 DEPTNO 值为 50 时，表达式的输出为：

```

not (deptno=10 or deptno=50 or deptno=null)
(false or false or null)
(false or null)
null

```

在 SQL 中, “TRUE or NULL” 的结果就是 TRUE, 而 “FALSE or NULL” 的结果是 NULL! 当在结果中有一个 NULL 值时, NULL 就会延续下去 (除非采用了跟 1.11 节中类似的技巧对 NULL 值作特别测试)。在使用 IN 谓词以及进行逻辑 OR 计算, 并且与 NULL 值有关的情况下, 这一点必须要记住。

要解决与 NOT IN 和 NULL 有关的问题, 可以使用 NOT EXISTS 和相关子查询。这里用了“相关子查询”这一术语, 因为子查询中要引用外部查询的行。下面的解决方案不受空值的影响 (请参考“问题”部分的内容):

```

select d.deptno
  from dept d
 where not exists ( select null
                    from emp e
                    where d.deptno = e.deptno )

DEPTNO
-----
40

```

从理论上讲, 在这种解决方案的外层查询中, 考虑到了表 DEPT 中的每一行。对于 DEPT 表中的每一行, 有以下几种情况:

1. 执行子查询, 查找部门编号是否在表 EMP 中, 注意条件 D.DEPTNO = E.DEPTNO, 该条件将两个表的部门编号合并到一起。
2. 如果子查询返回结果, 那么, EXISTS() 的值为真, NOT EXISTS(...) 的结果为 FALSE, 那么, 从外层查询中得到的该行就被舍弃掉。
3. 如果子查询没有返回结果, 则 NOT EXISTS(...) 的值为 TRUE, 而外层查询中得到的该行将被返回 (因为该部门并没有在 EMP 表中)。

EXISTS/NOT EXISTS 与相关子查询一起使用的时候, 子查询 SELECT 列表中的项目并不重要, 因此, 这里选择了 NULL, 把重点放在联接子查询上, 而不是 SELECT 列表的项目。

### 3.5 在一个表中查找与其他表不匹配的记录问题

对于具有相同关键字的两个表, 要在一个表中查找与另外一个表中不匹配的行。例如, 要查找没有职员部门的, 结果集应如下所示:

DEPTNO	DNAME	LOC
40	OPERATIONS	BOSTON

要查找部门中每个员工的工作岗位需要在表 DEPTNO 及 EMP 中有一个等值联接。  
www.TopSage.com

DEPTNO 列就是这两个表之间的公共值。但是，等值联接却不能直接显示出那个部门没有员工。这是因为在表 EMP 和 DEPT 正在等值联接时，将会返回满足联接条件的所有行。可是，我们只需要那些在表 DEPT 中不满足联接条件的的行。

尽管乍看起来这个问题同前一个问题类似，但是要复杂一些。其不同之处就是，在前一个问题中，需要列出在表 DEPT 中找出在表 EMP 中没有的部门编号。本节要球可以直接列出 DEPT 表中其他的列，而不仅仅是部门编号。

## 解决方案

首先返回一个表中的所有行，以及另一个表中与公共列匹配的行或不存在匹配行，然后，仅保留不匹配的行。

### DB2, MySQL, PostgreSQL, SQL Server

使用外联接及 NULL 筛选（OUTER 关键字是可选的）：

```
1 select d.*
2   from dept d left outer join emp e
3     on (d.deptno = e.deptno)
4  where e.deptno is null
```

### Oracle

在 Oracle9i Database 及其以后的版本中也可以使用前一解决方案，也可以使用 Oracle 特有的外联接语法。

```
1 select d.*
2   from dept d, emp e
3  where d.deptno = e.deptno (+)
4     and e.deptno is null
```

在 Oracle8i 及其以前的版本中，只能使用这种独有的外联接语法（注意在括号中的“+”号）。

## 讨论

这种解决方案使用外联接，然后只保留不匹配的记录。这种操作有时也被称为反联接。要想更好地了解反联接的工作机理，首先看看下面的这个尚未筛选空值的结果集：

```
select e.ename, e.deptno as emp_deptno, d.*
  from dept d left join emp e
    on (d.deptno = e.deptno)
```

ENAME	EMP_DEPTNO	DEPTNO	DNAME	LOC
SMITH	20	20	RESEARCH	DALLAS
ALLEN	30	30	SALES	CHICAGO
WARD	30	30	SALES	CHICAGO
JONES	20	20	RESEARCH	DALLAS
MARTIN	30	30	SALES	CHICAGO
BLAKE	30	30	SALES	CHICAGO
CLARK	10	10	ACCOUNTING	NEW YORK
SCOTT	20	20	RESEARCH	DALLAS

KING	10	10 ACCOUNTING	NEW YORK
TURNER	30	30 SALES	CHICAGO
ADAMS	20	20 RESEARCH	DALLAS
JAMES	30	30 SALES	CHICAGO
FORD	20	20 RESEARCH	DALLAS
MILLER	10	10 ACCOUNTING	NEW YORK
		40 OPERATIONS	BOSTON

注意，表 EMP 中最后一行的 ENAME 和 EMP\_DEPTNO 字段的值为 NULL。这是因为在部门 40 中没有员工。此解决方案使用 WHERE 子句只保留 EMP\_DEPTNO 字段值为 NULL 的行（也就是，在表 DEPT 的行中仅保留表 EMP 不存在匹配行的部分）。

### 3.6 向查询中增加联接而不影响其他联接问题

已经有了一个查询可以返回所需要的值，还需要得到其他的信息，但当加入这些信息时，发现原始结果集中的数据有丢失。例如，要返回所有的员工信息、他们工作部门的地点及所获得的奖励。在这个问题中，表 EMP\_BONUS 包含如下内容：

```
select * from emp_bonus
```

EMPNO	RECEIVED	TYPE
7369	14-MAR-2005	1
7900	14-MAR-2005	2
7788	14-MAR-2005	3

开始的查询如下所示：

```
select e.ename, d.loc
  from emp e, dept d
 where e.deptno=d.deptno
```

ENAME	LOC
SMITH	DALLAS
ALLEN	CHICAGO
WARD	CHICAGO
JONES	DALLAS
MARTIN	CHICAGO
BLAKE	CHICAGO
CLARK	NEW YORK
SCOTT	DALLAS
KING	NEW YORK
TURNER	CHICAGO
ADAMS	DALLAS
JAMES	CHICAGO
FORD	DALLAS
MILLER	NEW YORK

现要将每个员工所获得的奖励的日期列加入到结果数据中，但是联接到 EMP\_BONUS 表后，所返回的记录数要比所希望的要少，因为并不是每个员工都有奖励：

```
select e.ename, d.loc, eb.received
  from emp e, dept d, emp_bonus eb
 where e.deptno=d.deptno
    and e.empno=eb.empno
```



ENAME	LOC	RECEIVED
SCOTT	DALLAS	14-MAR-2005
SMITH	DALLAS	14-MAR-2005
JAMES	CHICAGO	14-MAR-2005

但实际上想要得到的结果集应如下所示：

ENAME	LOC	RECEIVED
ALLEN	CHICAGO	
WARD	CHICAGO	
MARTIN	CHICAGO	
JAMES	CHICAGO	14-MAR-2005
TURNER	CHICAGO	
BLAKE	CHICAGO	
SMITH	DALLAS	14-MAR-2005
FORD	DALLAS	
ADAMS	DALLAS	
JONES	DALLAS	
SCOTT	DALLAS	14-MAR-2005
CLARK	NEW YORK	
KING	NEW YORK	
MILLER	NEW YORK	

## 解决方案

可以使用外联接来获得这些附加的信息，并且原始查询中的数据也不会丢失。首先，联接表 EMP 和 DEPT 来得到所有员工姓名和他们工作的部门，然后与表 EMP\_BONUS 进行外联接来返回获得奖励的日期（对获得奖励的员工）。在 DB2、MySQL、PostgreSQL 和 SQL Server 中，语法如下所示：

```

1 select e.ename, d.loc, eb.received
2   from emp e join dept d
3     on (e.deptno=d.deptno)
4  left join emp_bonus eb
5     on (e.empno=eb.empno)
6  order by 2
```

在 Oracle9i Database 或其以后的版本中也可以使用上述的解决方案。另外，还可以使用 Oracle 的独有的外部联接语法，而在 Oracle8i Database 及其早期版本中，则只能使用下面列出的这种方案：

```

1 select e.ename, d.loc, eb.received
2   from emp e, dept d, emp_bonus eb
3  where e.deptno=d.deptno
4     and e.empno=eb.empno (+)
5  order by 2
```

还可以使用标量子查询（放在 SELECT 列表中的子查询）来模仿外联接：

```

1 select e.ename, d.loc,
2       (select eb.received from emp_bonus eb
3        where eb.empno=e.empno) as received
4   from emp e, dept d
5  where e.deptno=d.deptno
6  order by 2
```

这种使用标量子查询的解决方案可以在所有的平台上运行。

## 讨论

外联接可以返回一个表中所有行及与另一个表中匹配的行，可以参看前一节有关外联接的另一个例子。因为使用外联接不会将没有结果的行清除，而是将其返回，所以使用外部联接可以解决这个问题。此查询会返回不用外联接时应返回的所有行，并且，如果存在新加的数据也一并返回。

使用标量子查询也是解决这种问题的一个方便的技巧，因为它不需要修改已有的正确联接。使用标量子查询是不会危及到当前结果集而获得额外数据的一种简单方法。当使用标量子查询时，必须确保返回的是标量值（单个值）。如果在SELECT列表中的子查询的返回值超过一行，将会出现错误。

## 参阅

第14章14.10节，这一部分就是讲述如何绕过SELECT列表中的子查询不能返回多值的问题。

## 3.7 检测两个表中是否有相同的数据问题

要知道两个表或视图中是否有相同的数据（基数和值）。考虑这个视图：

```
create view v
as
select * from emp where deptno != 10
union all
select * from emp where ename = 'WARD'
select * from v
```

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7369	SMITH	CLERK	7902	17-DEC-1980	800		20
7499	ALLEN	SALESMAN	7698	20-FEB-1981	1600	300	30
7521	WARD	SALESMAN	7698	22-FEB-1981	1250	500	30
7566	JONES	MANAGER	7839	02-APR-1981	2975		20
7654	MARTIN	SALESMAN	7698	28-SEP-1981	1250	1400	30
7698	BLAKE	MANAGER	7839	01-MAY-1981	2850		30
7788	SCOTT	ANALYST	7566	09-DEC-1982	3000		20
7844	TURNER	SALESMAN	7698	08-SEP-1981	1500	0	30
7876	ADAMS	CLERK	7788	12-JAN-1983	1100		20
7900	JAMES	CLERK	7698	03-DEC-1981	950		30
7902	FORD	ANALYST	7566	03-DEC-1981	3000		20
7521	WARD	SALESMAN	7698	22-FEB-1981	1250	500	30

现要检测这个视图与表EMP中的数据是否完全相同。员工“WARD”行重复，说明解决方案不仅要显示不同行，还要显示重复行。因为在表EMP中部门10中的员工有3行，而对于员工“WARD”来说有2行。要返回下列的结果集应为：

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO	CNT
7521	WARD	SALESMAN	7698	22-FEB-1981	1250	500	30	1
7521	WARD	SALESMAN	7698	22-FEB-1981	1250	500	30	2

7782	CLARK	MANAGER	7839	09-JUN-1981	2450	10	1
7839	KING	PRESIDENT		17-NOV-1981	5000	10	1
7934	MILLER	CLERK	7782	23-JAN-1982	1300	10	1

## 解决方案

使用差集函数（MINUS 或 EXCEPT 根据 DBMS 的不同而选择）来解决这种表之间比较的问题是一种简单的解决办法。如果所使用的 DBMS 不支持此类函数，还可以使用相关子查询。

## DB2 和 PostgreSQL

使用集合运算函数 EXCEPT 和 UNION ALL 求视图 V 和表 EMP 的差集，以及表 EMP 和视图 V 的差集，并将两个差集合并：

```

1  (
2  select empno,ename,job,mgr,hiredate,sal,comm,deptno,
3         count(*) as cnt
4         from V
5         group by empno,ename,job,mgr,hiredate,sal,comm,deptno
6  except
7  select empno,ename,job,mgr,hiredate,sal,comm,deptno,
8         count(*) as cnt
9         from emp
10        group by empno,ename,job,mgr,hiredate,sal,comm,deptno
11 )
12 union all
13 (
14 select empno,ename,job,mgr,hiredate,sal,comm,deptno,
15        count(*) as cnt
16        from emp
17        group by empno,ename,job,mgr,hiredate,sal,comm,deptno
18  except
19  select empno,ename,job,mgr,hiredate,sal,comm,deptno,
20        count(*) as cnt
21        from v
22        group by empno,ename,job,mgr,hiredate,sal,comm,deptno
23 )

```

## Oracle

使用集合运算函数 MINUS 和 UNION ALL 求视图 V 和表 EMP 的差集，以及表 EMP 和视图 V 的差集，并将两个差集合并：

```

1  (
2  select empno,ename,job,mgr,hiredate,sal,comm,deptno,
3         count(*) as cnt
4         from V
5         group by empno,ename,job,mgr,hiredate,sal,comm,deptno
6  minus
7  select empno,ename,job,mgr,hiredate,sal,comm,deptno,
8         count(*) as cnt
9         from emp
10        group by empno,ename,job,mgr,hiredate,sal,comm,deptno
11 )
12 union all
13 (
14 select empno,ename,job,mgr,hiredate,sal,comm,deptno,
15        count(*) as cnt
16        from emp

```

```

17  group by empno,ename,job,mgr,hiredate,sal,comm,deptno
18  minus
19  select empno,ename,job,mgr,hiredate,sal,comm,deptno,
20         count(*) as cnt
21  from v
22  group by empno,ename,job,mgr,hiredate,sal,comm,deptno
23  )

```

## MySQL 和 SQL Server

使用关联子查询和 UNION ALL 来查找在视图 V 中存在而在表 EMP 中不存在的行，然后与在表 EMP 中存在而在视图 V 中不存在的行进行合并：

```

1  select *
2  from (
3  select e.empno,e.ename,e.job,e.mgr,e.hiredate,
4         e.sal,e.comm,e.deptno, count(*) as cnt
5  from emp e
6  group by empno,ename,job,mgr,hiredate,
7         sal,comm,deptno
8  ) e
9  where not exists (
10 select null
11 from (
12 select v.empno,v.ename,v.job,v.mgr,v.hiredate,
13        v.sal,v.comm,v.deptno, count(*) as cnt
14 from v
15 group by empno,ename,job,mgr,hiredate,
16         sal,comm,deptno
17 ) v
18 where v.empno = e.empno
19 and v.ename = e.ename
20 and v.job = e.job
21 and v.mgr = e.mgr
22 and v.hiredate = e.hiredate
23 and v.sal = e.sal
24 and v.deptno = e.deptno
25 and v.cnt = e.cnt
26 and coalesce(v.comm,0) = coalesce(e.comm,0)
27 )
28 union all
29 select *
30 from (
31 select v.empno,v.ename,v.job,v.mgr,v.hiredate,
32        v.sal,v.comm,v.deptno, count(*) as cnt
33 from v
34 group by empno,ename,job,mgr,hiredate,
35         sal,comm,deptno
36 ) v
37 where not exists (
38 select null
39 from (
40 select e.empno,e.ename,e.job,e.mgr,e.hiredate,
41        e.sal,e.comm,e.deptno, count(*) as cnt
42 from emp e
43 group by empno,ename,job,mgr,hiredate,
44         sal,comm,deptno
45 ) e
46 where v.empno = e.empno
47 and v.ename = e.ename
48 and v.job = e.job
49 and v.mgr = e.mgr
50 and v.hiredate = e.hiredate
51 and v.sal = e.sal
52 and v.deptno = e.deptno
53 and v.cnt = e.cnt

```

```

54      and coalesce(v.comm,0) = coalesce(e.comm,0)
55  )

```

## 讨论

不管使用那种技术方案，所有解决方案的原理都是一样的：

1. 首先，查找出表 EMP 中存在而视图 V 中没有的行。
2. 然后合并 (UNION ALL) 在视图 V 中存在，而在表 EMP 中没有的行。

如果在问题中的表是相等的，那么将没有数据行返回；如果这两个表不相同，则返回有差异的行。因为初次涉及对表作比较，单独比较基数要比包含数据的比较简单一些。下面列出的查询就是这样的一个简单例子，适用于所有 DBMS。

```

select count(*)
  from emp
 union
select count(*)
  from dept

COUNT(*)
-----
         4
        14

```

因为使用 UNION 可以筛选出重复行，所以如果两个表的基数相同，则只会返回一行。因为在这个例子中返回了两行，所以可以知道这两个表的行数不同。

## DB2、Oracle 和 PostgreSQL

MINUS 和 EXCEPT 的操作方法一样，所以在这里使用 EXCEPT 进行讨论。在 UNION ALL 之前和之后的查询非常相似。所以，要理解这种解决方案机理，只需单独执行一下 UNION ALL 之前的查询。下面的结果集是执行解决方案中 1~11 行后得到的：

```

(
  select empno,ename,job,mgr,hiredate,sal,comm,deptno,
         count(*) as cnt
    from v
   group by empno,ename,job,mgr,hiredate,sal,comm,deptno
 except
  select empno,ename,job,mgr,hiredate,sal,comm,deptno,
         count(*) as cnt
    from emp
   group by empno,ename,job,mgr,hiredate,sal,comm,deptno
)

```

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO	CNT
7521	WARD	SALESMAN	7698	22-FEB-1981	1250	500	30	2

此结果集表示，视图 V 中有一行，要么在表 EMP 中不存在同一行，要么在表 EMP 中同一行的基数不同。本例的是发现员工“WARD”重复，因而返回该行。如果仍然难以理解结果集是如何产生的，可以分别运行 EXCEPT 子句两边的查询，可以发现这两个结果集之间的唯一区别只是视图 V 中员工“WARD”的 CNT 不同。

在UNION ALL后面的查询所做的操作与UNION ALL之前的查询相反,它返回在表EMP中存在,而视图V中所没有的行。

```
(
  select empno,ename,job,mgr,hiredate,sal,comm,deptno,
         count(*) as cnt
    from emp
   group by empno,ename,job,mgr,hiredate,sal,comm,deptno
 minus
  select empno,ename,job,mgr,hiredate,sal,comm,deptno,
         count(*) as cnt
    from v
   group by empno,ename,job,mgr,hiredate,sal,comm,deptno
)
```

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO	CNT
7521	WARD	SALESMAN	7698	22-FEB-1981	1250	500	30	1
7782	CLARK	MANAGER	7839	09-JUN-1981	2450		10	1
7839	KING	PRESIDENT		17-NOV-1981	5000		10	1
7934	MILLER	CLERK	7782	23-JAN-1982	1300		10	1

用UNION ALL将两个结果集合并起来就得到最后的结果集。

## MySQL和SQL Server

UNION ALL之前和之后的查询十分相似。要理解这种基于子查询的解决方案的机理,只需要单独执行UNION ALL之前的查询即可。下面的查询是在解决方案中的1~27行:

```
select *
  from (
    select e.empno,e.ename,e.job,e.mgr,e.hiredate,
           e.sal,e.comm,e.deptno, count(*) as cnt
      from emp e
     group by empno,ename,job,mgr,hiredate,
              sal,comm,deptno
    ) e
 where not exists (
  select null
    from (
    select v.empno,v.ename,v.job,v.mgr,v.hiredate,
           v.sal,v.comm,v.deptno, count(*) as cnt
      from v
     group by empno,ename,job,mgr,hiredate,
              sal,comm,deptno
    ) v
   where v.empno = e.empno
        and v.ename = e.ename
        and v.job = e.job
        and v.mgr = e.mgr
        and v.hiredate = e.hiredate
        and v.sal = e.sal
        and v.deptno = e.deptno
        and v.cnt = e.cnt
        and coalesce(v.comm,0) = coalesce(e.comm,0)
  )
)
```

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO	CNT
7521	WARD	SALESMAN	7698	22-FEB-1981	1250	500	30	1
7782	CLARK	MANAGER	7839	09-JUN-1981	2450		10	1
7839	KING	PRESIDENT		17-NOV-1981	5000		10	1
7934	MILLER	CLERK	7782	23-JAN-1982	1300		10	1

注意，这里不是在比较表 EMP 和视图 V，而是在比较内联视图 E 和内联视图 V。内联视图求出每一行的基数，并作为该行的属性返回，因此要比较的是每一行的数据以及它们出现的次数。如果无法理解这种比较的机理，可以单独运行其中的子查询。下一步是在内联视图 E 中找出所有的在内联视图 V 中没有的行（包括 CNT）。比较操作使用了一个相关子查询和 NOT EXISTS 子句。联接操作可以判别那些行是相同的，结果是在内联视图 E 中存在，而没有包含在联接结果中的所有行。UNION ALL 后面的查询所做的操作与之相反，将查找所有在内联视图 V 中存在，而在内联视图 E 中没有的行：

```
select *
  from (
select v.empno,v.ename,v.job,v.mgr,v.hiredate,
      v.sal,v.comm,v.deptno, count(*) as cnt
  from v
 group by empno,ename,job,mgr,hiredate,
          sal,comm,deptno
        ) v
 where not exists (
select null
  from (
select e.empno,e.ename,e.job,e.mgr,e.hiredate,
      e.sal,e.comm,e.deptno, count(*) as cnt
  from emp e
 group by empno,ename,job,mgr,hiredate,
          sal,comm,deptno
        ) e
 where v.empno    = e.empno
       and v.ename = e.ename
       and v.job   = e.job
       and v.mgr   = e.mgr
       and v.hiredate = e.hiredate
       and v.sal   = e.sal
       and v.deptno = e.deptno
       and v.cnt   = e.cnt
       and coalesce(v.comm,0) = coalesce(e.comm,0)
  )
)
```

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO	CNT
7521	WARD	SALESMAN	7698	22-FEB-1981	1250	500	30	2

用 UNION ALL 将两个结果集合并起来就得到最后的结果集。

---

注意：Ales Spector 和 Jonathan Gennick 在“Transact-SQL Cookbook (O'Reilly)”一书中给出了另一种解决方案，参看其第 2 章中的“Comparing Two Sets for Equality”一节。

---

## 3.8 识别和消除笛卡儿积问题

要返回在部门 10 中每个员工的姓名，以及部门的工作地点，下面的查询得到的是错误数据：

```
select e.ename, d.loc
  from emp e, dept d
 where e.deptno = 10

ENAME      LOC
```

```

-----
CLARK      NEW YORK
CLARK      DALLAS
CLARK      CHICAGO
CLARK      BOSTON
KING       NEW YORK
KING       DALLAS
KING       CHICAGO
KING       BOSTON
MILLER     NEW YORK
MILLER     DALLAS
MILLER     CHICAGO
MILLER     BOSTON

```

正确的结果集应当如下所示：

```

-----
ENAME      LOC
-----
CLARK      NEW YORK
KING       NEW YORK
MILLER     NEW YORK

```

## 解决方案

在 FROM 子句对表进行联接来返回正确的结果集：

```

1 select e.ename, d.loc
2   from emp e, dept d
3  where e.deptno = 10
4        and d.deptno = e.deptno

```

## 讨论

请看表 DEPT 中的数据：

```

select * from dept
-----
DEPTNO DNAME      LOC
-----
10 ACCOUNTING NEW YORK
20 RESEARCH   DALLAS
30 SALES       CHICAGO
40 OPERATIONS BOSTON

```

可以看出，部门10的工作地点是在New York，所以，在返回值中部门所在地点除了NEW YORK以外的任何值都是错误的。错误查询得到的行数是FROM子句后面两个表基数的积。在原查询中，对表EMP的筛选条件是部门为10，结果有3行；因为没有对表DEPT进行筛选，表DEPT的所有4行全部返回，3乘以4得12，所以这个错误查询就返回了12行。一般来说，要避免产生笛卡儿积，需要使用 $n-1$ 规则，这里的 $n$ 为FROM子句中表的数量，并且， $n-1$ 是要避免产生笛卡儿积的最小联接数。根据在表中的关键字和联接列不同，可能需要超过 $n-1$ 个联接，但是对当写查询来说， $n-1$ 是一个好的开始。

---

**注意：**如果笛卡儿积应用适当也很有用。很多查询都用到了笛卡儿积，常用的场合有转置（反向转置）结果集、产生顺序值和模拟循环等。

---



### 3.9 聚集与联接

#### 问题

要在包含多个表的查询中执行聚集运算，要确保表间联接不能使聚集运算发生错误。例如，要查找在部门10中所有员工的工资合计和奖金合计。由于有些员工的奖金记录不止一条，在表EMP和表EMP\_BONUS之间做联接会导致聚集函数SUM算得的值错误。在此问题中，表EMP\_BONUS包含如下数据：

```
select * from emp_bonus
```

EMPNO	RECEIVED	TYPE
7934	17-MAR-2005	1
7934	15-FEB-2005	2
7839	15-FEB-2005	3
7782	15-FEB-2005	1

现在，考虑一下下面返回的在部门10中所有员工的工资和奖金的查询。表BONUS中的TYPE字段决定奖金额，类型1的奖金为员工工资的10%，类型2为20%，类型3为30%。

```
select e.empno,
       e.ename,
       e.sal,
       e.deptno,
       e.sal*case when eb.type = 1 then .1
                  when eb.type = 2 then .2
                  else .3
                end as bonus
from emp e, emp_bonus eb
where e.empno = eb.empno
and e.deptno = 10
```

EMPNO	ENAME	SAL	DEPTNO	BONUS
7934	MILLER	1300	10	130
7934	MILLER	1300	10	260
7839	KING	5000	10	1500
7782	CLARK	2450	10	245

进行到这儿，一切正常。然而，为了计算奖金总数而跟表EMP\_BONUS做联接时，错误出现了：

```
select deptno,
       sum(sal) as total_sal,
       sum(bonus) as total_bonus
from (
select e.empno,
       e.ename,
       e.sal,
       e.deptno,
       e.sal*case when eb.type = 1 then .1
                  when eb.type = 2 then .2
                  else .3
                end as bonus
from emp e, emp_bonus eb
where e.empno = eb.empno
and e.deptno = 10
) x
group by deptno
```

DEPTNO	TOTAL_SAL	TOTAL_BONUS
10	10050	2135

尽管TOTAL\_BONUS所返回的值是正确的，TOTAL\_SAL却是错误的，下面的查询结果表明，部门10的工资总数应当为8750：

```
select sum(sal) from emp where deptno=10
SUM(SAL)
-----
8750
```

TOTAL\_SAL为什么错了？因为联接导致SAL列存在重复。考虑下面的查询，该查询联接表EMP和EMP\_BONUS：

```
select e.ename,
       e.sal
  from emp e, emp_bonus eb
 where e.empno = eb.empno
       and e.deptno = 10
```

ENAME	SAL
CLARK	2450
KING	5000
MILLER	1300
MILLER	1300

现在可以很容易地看出TOTAL\_SAL为什么错了，因为MILLER的工资被统计了两次。真正想要的结果集应当如下所示：

DEPTNO	TOTAL_SAL	TOTAL_BONUS
10	8750	2135

## 解决方案

当处理聚集与联接混合操作时，一定要小心。如果联接产生重复行，可以有两种方法来避免聚集函数计算错误：方法之一，只要在调用聚集函数时使用关键字DISTINCT，这样每个值只参与计算一次；另一种方法是，在进行联接前先执行聚集操作（在内联视图中），这样，因为聚集计算已经在进行联接前完成了，所以就可以避免聚集函数计算错误，从而可以完全避免产生此问题。下面列出的解决方案使用了DISTINCT关键字，而“讨论”部分将讨论如何在联接前使用内联视图来执行聚集操作。

## MySQL 和 PostgreSQL

使用DISTINCT关键字只对不相同的工资求和：

```
1 select deptno,
2        sum(distinct sal) as total_sal,
3        sum(bonus) as total_bonus
4  from (
5  select e.empno,
6         e.ename,
7         e.sal,
8         e.deptno,
9         e.sal*case when eb.type = 1 then .1
```

```

10             when eb.type = 2 then .2
11             else .3
12         end as bonus
13     from emp e, emp_bonus eb
14     where e.empno = eb.empno
15           and e.deptno = 10
16           ) x
17     group by deptno

```

## DB2、Oracle 和 SQL Server

这些平台也支持上面的解决方案，此外它们还支持另一种使用窗口函数 SUM OVER 方案：

```

1  select distinct deptno,total_sal,total_bonus
2      from (
3  select e.empno,
4         e.ename,
5         sum(distinct e.sal) over
6         (partition by e.deptno) as total_sal,
7         e.deptno,
8         sum(e.sal*case when eb.type = 1 then .1
9                     when eb.type = 2 then .2
10                    else .3 end) over
11         (partition by deptno) as total_bonus
12     from emp e, emp_bonus eb
13     where e.empno = eb.empno
14           and e.deptno = 10
15           ) x

```

## 讨论

### MySQL 和 PostgreSQL

在“问题”部分的第二个查询在联接表 EMP 和 EMP\_BONUS 时，对员工“MILLER”产生了两条记录，这就是导致计算 EMP.SAL 的和出错的原因（其工资加了两次）。解决方案是只把不同的 EMP.SAL 值相加。下面的查询是另外一种解决方法，首先计算部门 10 中工资合计，然后将该行跟表 EMP 联接，最后联接到表 EMP\_BONUS。下面的查询可以用于所有 DBMS：

```

select d.deptno,
       d.total_sal,
       sum(e.sal*case when eb.type = 1 then .1
                   when eb.type = 2 then .2
                   else .3 end) as total_bonus
  from emp e,
       emp_bonus eb,
       (
select deptno, sum(sal) as total_sal
  from emp
 where deptno = 10
 group by deptno
       ) d
 where e.deptno = d.deptno
       and e.empno = eb.empno
 group by d.deptno,d.total_sal

```

DEPTNO	TOTAL_SAL	TOTAL_BONUS
10	8750	2135

## DB2、Oracle 和 SQL Server

另一种解决方案发挥了窗口函数 SUM OVER 的优势。下面的查询是该“解决方案”中的第 3~14 行，并且返回下面的结果集：

```
select e.empno,
       e.ename,
       sum(distinct e.sal) over
         (partition by e.deptno) as total_sal,
       e.deptno,
       sum(e.sal*case when eb.type = 1 then .1
                    when eb.type = 2 then .2
                    else .3 end) over
         (partition by deptno) as total_bonus
  from emp e, emp_bonus eb
 where e.empno = eb.empno
        and e.deptno = 10
```

EMPNO	ENAME	TOTAL_SAL	DEPTNO	TOTAL_BONUS
7934	MILLER	8750	10	2135
7934	MILLER	8750	10	2135
7782	CLARK	8750	10	2135
7839	KING	8750	10	2135

在上述查询中，SUM OVER 窗口函数被两次调用，第一次用来计算给定分区或组中不同工资合计，本例中，分区为 DEPTNO 为 10，部门 10 不相同工资的总额为 8750；第二次调用 SUM OVER 用来计算同一分区的奖金合计。取 TOTAL\_SAL、DEPT\_NO 和 TOTAL\_BONUS 的唯一值就得到最终结果集。

## 3.10 聚集与外联接

### 问题

问题的开始跟 3.9 节一样，区别在于，修改 EMP\_BONUS 表，使得在部门 10 中并不是每个的员工都有奖金。考虑如下的 EMP\_BONUS 表和（表面上）计算出部门 10 中所有员工的工资总额及其奖金总额的查询。

```
select * from emp_bonus
```

EMPNO	RECEIVED	TYPE
7934	17-MAR-2005	1
7934	15-FEB-2005	2

```
select deptno,
       sum(sal) as total_sal,
       sum(bonus) as total_bonus
  from (
select e.empno,
       e.ename,
       e.sal,
       e.deptno,
       e.sal*case when eb.type = 1 then .1
                  when eb.type = 2 then .2
                  else .3 end as bonus
  from emp e, emp_bonus eb
 where e.empno = eb.empno
        and e.deptno = 10
```

```

    )
  group by deptno
DEPTNO  TOTAL_SAL  TOTAL_BONUS
-----
    10         2600         390

```

TOTAL\_BONUS 的结果是正确的, 但是 TOTAL\_SAL 的值却不能体现在部门 10 中所有员工的工资。下面的查询可揭示 TOTAL\_SAL 不正确的原因:

```

select e.empno,
       e.ename,
       e.sal,
       e.deptno,
       e.sal*case when eb.type = 1 then .1
                  when eb.type = 2 then .2
                  else .3 end as bonus
from emp e, emp_bonus eb
where e.empno = eb.empno
      and e.deptno = 10

```

EMPNO	ENAME	SAL	DEPTNO	BONUS
7934	MILLER	1300	10	130
7934	MILLER	1300	10	260

与其说是汇总了部门 10 中所有员工的工资, 不如说是只汇总了“MILLER”一个人的工资, 并且还被错误地汇总了两次。实际上, 应该得到如下的结果集:

```

DEPTNO  TOTAL_SAL  TOTAL_BONUS
-----
    10         8750         390

```

## 解决方案

此问题的解决方案与 3.9 中的解决方案类似, 但是这里应当对表 EMP\_BONUS 使用外联接来确保在部门 10 中所有员工都包含到结果中:

## DB2、MySQL、PostgreSQL 和 SQL Server

外联接到表 EMP\_BONUS, 然后只对部门 10 中不同的工资进行汇总:

```

1  select deptno,
2         sum(distinct sal) as total_sal,
3         sum(bonus) as total_bonus
4  from (
5  select e.empno,
6         e.ename,
7         e.sal,
8         e.deptno,
9         e.sal*case when eb.type is null then 0
10                  when eb.type = 1 then .1
11                  when eb.type = 2 then .2
12                  else .3 end as bonus
13  from emp e left outer join emp_bonus eb
14    on (e.empno = eb.empno)
15  where e.deptno = 10
16  )
17  group by deptno

```

也可以使用窗口函数 SUM OVER:

```

1 select distinct deptno,total_sal,total_bonus
2   from (
3 select e.empno,
4        e.ename,
5        sum(distinct e.sal) over
6          (partition by e.deptno) as total_sal,
7        e.deptno,
8        sum(e.sal*case when eb.type is null then 0
9                      when eb.type = 1 then .1
10                     when eb.type = 2 then .2
11                     else .3
12                  end) over
13          (partition by deptno) as total_bonus
14   from emp e left outer join emp_bonus eb
15    on (e.empno = eb.empno)
16  where e.deptno = 10
17 ) x

```

## Oracle

如果使用 Oracle9i Database 及其以后的版本，可以使用上述的解决方案。另一种方案是使用 Oracle 特有的外联接语法，如果使用 Oracle8i 及其以前的版本，则只能使用这种语法：

```

1 select deptno,
2        sum(distinct sal) as total_sal,
3        sum(bonus) as total_bonus
4   from (
5 select e.empno,
6        e.ename,
7        e.sal,
8        e.deptno,
9        e.sal*case when eb.type is null then 0
10                  when eb.type = 1 then .1
11                  when eb.type = 2 then .2
12                  else .3 end as bonus
13   from emp e, emp_bonus eb
14  where e.empno = eb.empno (+)
15        and e.deptno = 10
16        )
17  group by deptno

```

Oracle 8i Database 用户也可以使用 DB2 和其他数据库的 SUM OVER 语法，但是必须像前一查询一样修改为 Oracle 特有的外联接语法。

## 讨论

在“问题”部分的第二个查询中，对表 EMP 和 EMP\_BONUS 作连接，结果只返回了员工“MILLER”的行，这就是导致对 EMP.SAL 求和错误地原因（DEPTNO 10 的其他员工没有奖金，他们的工资也没有计入工资总额中）。该解决方案将表 EMP 外联接到表 EMP\_BONUS，使得每个没有奖金的员工也包括在结果中。如果某个员工没有奖金，EMP\_BONUS.TYPE 将返回 NULL。一定要记住，这里对 CASE 语句做了修改，跟 3.9 节方案中有所不同，如果 EMP\_BONUS.TYPE 为 NULL，CASE 表达式返回 0，对汇总结果没有影响。

下面的查询是另一种解决方案。首先计算部门10的工资总额，然后联接到表EMP，最后再联接到表EMP\_BONUS（避免了外联接）。下面的查询适用于所有DBMS：

```
select d.deptno,
       d.total_sal,
       sum(e.sal*case when eb.type = 1 then .1
                  when eb.type = 2 then .2
                  else .3 end) as total_bonus
  from emp e,
       emp_bonus eb,
       (
select deptno, sum(sal) as total_sal
  from emp
 where deptno = 10
 group by deptno
 ) d
 where e.deptno = d.deptno
       and e.empno = eb.empno
 group by d.deptno,d.total_sal
```

DEPTNO	TOTAL_SAL	TOTAL_BONUS
10	8750	390

### 3.11 从多个表中返回丢失的数据

#### 问题

同时返回多个表中丢失的数据。要从表DEPT中返回EMP不存在的行（所有没有员工的部门）需要做外联接。考虑下面的查询，它返回表DEPT中的DEPTNO和DNAME字段，以及每个部门中所有员工的姓名（如果该某个部门有员工的话）：

```
select d.deptno,d.dname,e.ename
  from dept d left outer join emp e
    on (d.deptno=e.deptno)
```

DEPTNO	DNAME	ENAME
20	RESEARCH	SMITH
30	SALES	ALLEN
30	SALES	WARD
20	RESEARCH	JONES
30	SALES	MARTIN
30	SALES	BLAKE
10	ACCOUNTING	CLARK
20	RESEARCH	SCOTT
10	ACCOUNTING	KING
30	SALES	TURNER
20	RESEARCH	ADAMS
30	SALES	JAMES
20	RESEARCH	FORD
10	ACCOUNTING	MILLER
40	OPERATIONS	

最后一行 OPERATIONS 部门，尽管该部门没有员工，还是返回了这一行，因为表EMP被外联接到了表DEPT。现在，假设有一个员工没有部门，那么如何得到在上述的结果集，并为该没有部门的员工返回一行呢？换句话说，要在同一个的查询中同时外联接到表EMP和DEPT。在创建新员工之后，开始可能会这么做：

```
insert into emp (empno,ename,job,mgr,hiredate,sal,comm,deptno)
select 1111,'YODA','JEDI',null,hiredate,sal,comm,null
from emp
where ename = 'KING'
```

```
select d.deptno,d.dname,e.ename
from dept d right outer join emp e
on (d.deptno=e.deptno)
```

DEPTNO	DNAME	ENAME
10	ACCOUNTING	MILLER
10	ACCOUNTING	KING
10	ACCOUNTING	CLARK
20	RESEARCH	FORD
20	RESEARCH	ADAMS
20	RESEARCH	SCOTT
20	RESEARCH	JONES
20	RESEARCH	SMITH
30	SALES	JAMES
30	SALES	TURNER
30	SALES	BLAKE
30	SALES	MARTIN
30	SALES	WARD
30	SALES	ALLEN
		YODA

使用外联接是想返回新创建的员工,但却将原结果集中的OPERATIONS部门丢掉了。最终的结果集应为员工 YODA 和部门 OPERATIONS 各返回一行,如下所示:

DEPTNO	DNAME	ENAME
10	ACCOUNTING	CLARK
10	ACCOUNTING	KING
10	ACCOUNTING	MILLER
20	RESEARCH	ADAMS
20	RESEARCH	FORD
20	RESEARCH	JONES
20	RESEARCH	SCOTT
20	RESEARCH	SMITH
30	SALES	ALLEN
30	SALES	BLAKE
30	SALES	JAMES
30	SALES	MARTIN
30	SALES	TURNER
30	SALES	WARD
40	OPERATIONS	
		YODA

## 解决方案

使用基于公共值的完全外联接来返回两个表中丢失的数据。

## DB2、MySQL、PostgreSQL 和 SQL Server

显式使用 FULL OUTER JOIN 命令返回两个表中丢失的行以及所有匹配的行:

```
1 select d.deptno,d.dname,e.ename
2   from dept d full outer join emp e
3     on (d.deptno=e.deptno)
```

或者,合并两个不同外联接的结果:



```

1 select d.deptno,d.dname,e.ename
2   from dept d right outer join emp e
3     on (d.deptno=e.deptno)
4 union
5 select d.deptno,d.dname,e.ename
6   from dept d left outer join emp e
7     on (d.deptno=e.deptno)

```

## Oracle

如果使用 Oracle9i Database 及其以后的版本, 上述的两种解决方案都可以使用。也可以使用 Oracle 所特有的外联接语法, 在 Oracle8i Database 及其以前的版本中, 只能应用此语法来解决上述问题。

```

1 select d.deptno,d.dname,e.ename
2   from dept d, emp e
3  where d.deptno = e.deptno(+)
4 union
5 select d.deptno,d.dname,e.ename
6   from dept d, emp e
7  where d.deptno(+) = e.deptno

```

## 讨论

完全外联接就是所有表的外联接简单地合并。要知道完全外联接的“详细内幕”, 只需运行每个外联接, 然后合并结果集即可。下面的查询返回表 DEPT 的所有行及表 EMP 中与之匹配的行 (如果有的话):

```

select d.deptno,d.dname,e.ename
  from dept d left outer join emp e
    on (d.deptno = e.deptno)

```

DEPTNO	DNAME	ENAME
20	RESEARCH	SMITH
30	SALES	ALLEN
30	SALES	WARD
20	RESEARCH	JONES
30	SALES	MARTIN
30	SALES	BLAKE
10	ACCOUNTING	CLARK
20	RESEARCH	SCOTT
10	ACCOUNTING	KING
30	SALES	TURNER
20	RESEARCH	ADAMS
30	SALES	JAMES
20	RESEARCH	FORD
10	ACCOUNTING	MILLER
40	OPERATIONS	

下面的查询返回表 EMP 的所有行及表 DEPT 中与之匹配的行 (如果有的话):

```

select d.deptno,d.dname,e.ename
  from dept d right outer join emp e
    on (d.deptno = e.deptno)

```

DEPTNO	DNAME	ENAME
10	ACCOUNTING	MILLER
10	ACCOUNTING	KING
10	ACCOUNTING	CLARK
20	RESEARCH	FORD

20	RESEARCH	ADAMS
20	RESEARCH	SCOTT
20	RESEARCH	JONES
20	RESEARCH	SMITH
30	SALES	JAMES
30	SALES	TURNER
30	SALES	BLAKE
30	SALES	MARTIN
30	SALES	WARD
30	SALES	ALLEN
		YODA

这两个查询合并的结果就是最终的结果集。

## 3.12 在运算和比较时使用 NULL 值

### 问题

NULL 值永远不会等于或不等于任何值，也包括 NULL 值自己，但是需要像计算真实值一样计算可为空列的返回值。例如，需要在表 EMP 中查出所有比“WARD”提成 (COMM) 低的员工，提成为 NULL (空) 的员工也应当包括在其中。

### 解决方案

使用 COALESCE 函数将 NULL 值转换为一个可以用来作为标准值进行比较的真实值：

```

1 select ename,comm
2   from emp
3  where coalesce(comm,0) < ( select comm
4                             from emp
5                             where ename = 'WARD' )

```

### 讨论

COALESCE 函数从值列表中返回第一个非 NULL 值。当遇到 NULL 值时将其替换为 0，这样就可以同 Ward 的提成进行比较了。也可以将 COALESCE 函数放置在 SELECT 列表中：

```

select ename,comm,coalesce(comm,0)
  from emp
 where coalesce(comm,0) < ( select comm
                           from emp
                           where ename = 'WARD' )

```

ENAME	COMM	COALESCE(COMM,0)
SMITH		0
ALLEN	300	300
JONES		0
BLAKE		0
CLARK		0
SCOTT		0
KING		0
TURNER	0	0
ADAMS		0
JAMES		0
FORD		0
MILLER		0

# 插入、更新与删除

前几章着重介绍了一些基本的查询技术，重点是如何从数据库中获取数据。本章要介绍以下三部分内容：

- 在数据库中插入新记录
- 更新已有记录
- 删除不需要的记录

为便于读者在需要的时候可以方便地查阅本章内容，本章按如下方式组织：首先是所有有关插入数据的内容，然后是更新部分，最后是删除数据部分。

通常，插入数据是一个简单的操作。首先从简单的插入单行的问题开始；然而很多时候采用基于集合的方法来创建新行更高效；最后，介绍如何一次插入多行的技术。

同样，更新和删除也从简单的任务开始。首先是更新或是删除一条记录，然后是使用更有效的方法来更新一大批记录。本章还介绍了使用很多方便的方法来删除记录，例如，可以根据是否在另外一个表中有相应的记录来删除源表中的记录。

对于 SQL 来说，还有另一种方法，可以一次进行插入、更新和删除记录的全部操作，这相对而言是标准中的新增功能。MERGE 语句目前似乎不十分有用，但是它代表了一种将数据库的表与外部数据源（例如来自远程系统的无格式文件）同步的十分强强有力的方法。有关细节请查看本章第 4.11 小节。

## 4.1 插入新记录

### 问题

向表中插入一条新的记录。例如，要向 DEPT 表中插入一条新的记录。其中，DEPTNO 值为 50、DNAME 的值为“PROGRAMMING”、LOC 的值为“BALTIMORE”。

## 解决方案

使用带有 VALUES 子句的 INSERT 语句来插入一行：

```
insert into dept (deptno,dname,loc)
values (50,'PROGRAMMING','BALTIMORE')
```

对于 DB2 和 MySQL，可以选择一次插入一行，或者用多个值列表一次插入多行：

```
/* multi row insert */
insert into dept (deptno,dname,loc)
values (1,'A','B'),
       (2,'B','C')
```

## 讨论

INSERT 语句允许在数据库表中创建新行。在所有类型的数据库系统中，插入语句的语法格式完全相同。

作为一种简便方式，在 INSERT 语句中，可以省略字段列表。

```
insert into dept
values (50,'PROGRAMMING','BALTIMORE')
```

然而，如果语句中没有列出要插入行中的目标字段，则必须要插入表中的所有列，需要注意的是，在插入值列表中所列出的值的顺序，必须与 SELECT \* 查询语句所列出的列顺序完全一致。

## 4.2 插入默认值

### 问题

定义表时可以为某些列定义默认值。现要以默认值插入一行，而无需指定各列的值。看一下下面列出的表：

```
create table D (id integer default 0)
```

要插入 0 值，而不想在 INSERT 语句的值列表中明确地为该行指定 0 值。这里明确要求插入默认值，而不管默认值是什么。

### 解决方案

所有数据库系统都支持使用 DEFAULT 关键字显式地指定某列插入默认值，有些数据库系统还有其他的方法来解决这个问题。

下面的例子说明了使用 DEFAULT 关键字来解决这个问题的方法：

```
insert into D values (default)
```

当不需要将表中所有列都插入值时，也可以明确地指定要使用默认值的列名称。

```
insert into D (id) values (default)
```

Oracle8i 数据库及其以前的版本不支持 DEFAULT 关键字。在 Oracle9i 以前版本的数据库中，无法显式地插入默认值。

在 MySQL 中，如果表中所有的列都定义了默认值，可以用一个空的值列表来解决此问题：

```
insert into D values ()
```

在这种情况下，所有的列将设置为其默认值。

PostgreSQL 和 SQL Server 支持 DEFAULT VALUES 子句：

```
insert into D default values
```

DEFAULT VALUES 子句将所有的列设置为其本身的默认值。

## 讨论

在值列表中的 DEFAULT 关键字为相应列插入默认值，默认值在创建表时定义。所有的 DBMS 中都可以使用此关键字。

如果表的每列都定义了默认值（表 D 就是这种情况），MySQL、PostgreSQL 和 SQL Server 用户也可以使用其他的方法。例如，可以使用空 VALUES 列表（MySQL）或者指定 DEFAULT VALUES 子句（PostgreSQL 和 SQL Server）来用默认值创建行；否则，就需要对表中的每一行指定 DEFAULT。

如果一个表部分列有默认值，部分列没有默认值，要向某列中插入默认值只要将该列排除在插入列表之外，都不需要使用 DEFAULT 关键字。假设表 D 有另外一列没有定义默认值：

```
create table D (id integer default 0, foo varchar(10))
```

在插入列表中只列出 FOO 字段，ID 中就可以插入默认值：

```
insert into D (name) values ('Bar')
```

使用上述语句的结果是 FOO 字段值为“Bar”，而 ID 字段的值为“0”。ID 字段的值为 0 是因为没有对其指定其他值。

## 4.3 使用 NULL 代替默认值

### 问题

在一个定义了默认值的列插入数据，并且需要不管该列的默认值是什么，都将该列值设为 NULL。考虑一下下面的表：

```
create table D (id integer default 0, foo VARCHAR(10))
```

希望插入一行，其中 ID 的值为 NULL。

## 解决方案

可以在值列表中明确地指定 NULL 值：

```
insert into d (id, foo) values (null, 'Brighten')
```

## 讨论

并不是所有的人都知道可以在 INSERT 语句的值列表中明确指定 NULL 值。人们通常的做法是，如果在插入操作时不需要对某列指定值，只是将该列从插入列表和值列表中去掉。

```
insert into d (foo) values ('Brighten')
```

在上面的语句中没有指定 ID 值。许多人可能认为列没有被指定值时，应当被赋予空值。但是，如果该列在创建的时候指定了默认值，结果就是在执行 INSERT 指令时，ID 字段的值被赋予 0（默认值）。如果想要将某列的值指定为 NULL，则需要指定该列值为 NULL，而不管该列是否有默认值。

## 4.4 从一个表向另外的表中复制行

### 问题

要使用查询从一个表中向另外的表中复制行。该查询可能非常复杂，也可能非常简单，但是最终是需要将该查询的结果插入到其他的表中。例如，要将表 DEPT 中的行复制到表 DEPT\_EAST 中。假定表 DEPT\_EAST 表已经创建，其结构与表 DEPT 相同（同样的列名称及数据类型），目前该表为空。

## 解决方案

所使用的方法就是在 INSERT 语句后面紧跟一个用来产生所要插入的行的查询：

```
1 insert into dept_east (deptno,dname,loc)
2 select deptno,dname,loc
3   from dept
4  where loc in ( 'NEW YORK','BOSTON' )
```

## 讨论

最简单的方法就是在 INSERT 语句后面跟随一个查询，该查询用来返回想要得到的行。如果想要从源表中复制所有的行，则在查询中就不需要使用 WHERE 子句。跟普通的插入语句一样，插入时不一定要指定插入哪些列，但是，如果没有指定所要插入的目标列，则必须插入表中所有的列，而且要清楚 SELECT 列表中值的顺序，就像在 4.1 节中所介绍的那样。

## 4.5 复制表定义

### 问题

要创建新表，该表与已有表的列设置相同。例如，想要创建一个 DEPT 表的副本，名为 DEPT\_2，但只是想复制表结构而不想要复制源表中的记录。

## 解决方案

### DB2

使用带有 LIKE 子句的 CREATE TABLE 命令：

```
create table dept_2 like dept
```

### Oracle、MySQL 和 PostgreSQL

在 CREATE TABLE 命令中，使用一个不返回任何行的子查询：

```
1 create table dept_2
2 as
3 select *
4   from dept
5  where 1 = 0
SQL Server
```

使用带有不返回任何行的查询（虽然原文中用的是 subquery，但这里跟一般所说的子查询不大相同，建议就改成查询，译者注）和 INTO 子句：

```
1 select *
2   into dept_2
3   from dept
4  where 1 = 0
```

## 讨论

### DB2

DB2 的 CREATE TABLE ... LIKE 命令应用简单，可以用一个表作为父表，来创建另外一个表。只需在 LIKE 关键字后面指定父表名称即可。

### Oracle、MySQL 和 PostgreSQL

当使用 Create Table As Select (CTAS) 语句时，除非在 WHERE 子句中指定了为 “false” 的条件（此时将生成一个空表），否则在查询中所生成的所有记录都会加到新生成的表中。在解决方案中，在 WHERE 子句中的表达式为 “1=0”，导致了查询不返回任何行，这样，CATS 语句的结果就是根据查询中 SELECT 子句中的列名而产生的空表。

### SQL Server

当使用 INTO 命令来复制表时，查询所返回的所有行都会加入新生成的表中，除非在 WHERE 子句中定义一个恒为 “false” 的条件。在解决方案中，在 WHERE 子句中的表达式为 “1=0”，导致了查询不返回任何行，这样，CATS 语句的结果就是根据查询中 SELECT 子句中的列名而产生的空表。

## 4.6 一次向多个表中插入记录

### 问题

要将一个查询中返回的行插入到多个目标表中。例如，要将表 DEPT 中的一些行插入到

表 DEPT\_EAST、DEPT\_WEST 和 DEPT\_MID 中。这三个表与表 DEPT 有着相同的结构（相同的列和数据类型），并且这三个表都是空的。

## 解决方案

此解决方案是要将查询的结果插入到目标表中。与 4.4 节不同的是，此问题中有多个目标表。

### Oracle

使用 INSERT ALL 或 INSERT FIRST 语句。这两种方法除了关键字 ALL 与 FIRST 不同外，其语法都相同。下面的语句使用了 INSERT ALL 命令来同时兼顾所有的目标表：

```
1 insert all
2   when loc in ('NEW YORK','BOSTON') then
3     into dept_east (deptno,dname,loc) values (deptno,dname,loc)
4   when loc = 'CHICAGO' then
5     into dept_mid (deptno,dname,loc) values (deptno,dname,loc)
6   else
7     into dept_west (deptno,dname,loc) values (deptno,dname,loc)
8   select deptno,dname,loc
9   from dept
```

### DB2

将所有目标表用 UNION ALL 构成一个内联视图，并以该内联视图作为 INSERT INTO 的目标。必须要在这些表中设置约束条件，以确保这些行插入到正确的表中。

```
create table dept_east
( deptno integer,
  dname varchar(10),
  loc varchar(10) check (loc in ('NEW YORK','BOSTON'))

create table dept_mid
( deptno integer,
  dname varchar(10),
  loc varchar(10) check (loc = 'CHICAGO'))

create table dept_west
( deptno integer,
  dname varchar(10),
  loc varchar(10) check (loc = 'DALLAS'))

1 insert into (
2   select * from dept_west union all
3   select * from dept_east union all
4   select * from dept_mid
5 ) select * from dept
```

### MySQL、PostgreSQL 和 SQL Server

在编写本书时，现有的所有版本都不支持这种多表插入操作。

## 讨论

### Oracle

Oracle 的多表插入操作使用 WHEN-THEN-ELSE 子句，判断嵌套的 SELECT 语句各结果行



的目标表，并插入到相应的表中。虽然在本章的示例中，使用 INSERT ALL 和 INSERT FIRST 将产生同样的结果，但是这两者之间还是有一定的区别。INSERT FIRST 在遇到条件表达式为真的情况时，将会立即跳出 WHEN-THEN-ELSE，而 INSERT ALL 即使前面的条件表达式为真，也会继续检查其他所有的条件，所以可以使用 INSERT ALL 命令来将同一行插入到多个表中。

## DB2

DB2 解决方案有些复杂。它需要所要插入数据的表有约束条件，以确保所有从子查询中返回的行将插入到正确的表中。解决方案所用的方法就是将目标表用 UNION ALL 定义成视图，并以它为插入目标。如果 INSERT 语句中各表之间的约束条件结果不唯一（也就是说多个表的约束条件结果相同），则 INSERT 语句就不知道把数据行放到哪个表中，造成插入操作失败。

## MySQL、PostgreSQL 和 SQL Server

到编写本书时，只有 Oracle 和 DB2 提供了用一条语句将查询返回的行插入到一个或多个表中的机制。

## 4.7 阻止对某几列插入问题

防止用户或是错误的软件应用程序对某几列插入数据。例如，只允许某个程序向 EMP 表中插入 EMPNO、ENAME 和 JOB 列。

### 解决方案

在表中创建一个视图，该视图将只显示允许用户进行操作的列，强制所有的插入操作都通过该视图进行。

例如，创建一个只显示表 EMP 中特定 3 列的视图：

```
create view new_emps as
select empno, ename, job
from emp
```

对只允许操作视图中三个字段的用户和程序，授权他们访问该视图，而不允许他们对表 EMP 进行插入操作。用户可以将新的记录插入到视图 NEW\_EMPS 中，从而创建 EMP 表中新的记录，但是不能对视图所定义的三个字段之外的列进行操作。

### 讨论

当对类似于本解决方案中的简单视图进行插入操作时，数据库服务器将插入操作转换到视图的基表。例如下面的插入语句：

```
insert into new_ems
(empno,ename,job)
values (1, 'Jonathan', 'Editor')
```

会被转换为以下列的插入语句：

```
insert into emp
(empno,ename,job)
values (1, 'Jonathan', 'Editor')
```

也可以，对内联视图进行插入（当前只有 Oracle 支持此项操作）：

```
insert into
(select empno, ename, job
 from emp)
values (1, 'Jonathan', 'Editor')
```

视图的插入操作是一个复杂的话题。除了最简单的视图之外，操作规则变得越来越复杂。如果想要用视图进行插入操作，必须要认真参考和理解相应数据库厂商的有关文档。

## 4.8 在表中编辑记录

### 问题

要修改表中某些（或全部）行的值。例如，可能想要将部门20中所有员工的工资增加10%。下面的结果集显示了该部门员工的 DEPTNO、ENAME 和 SAL 字段：

```
select deptno,ename,sal
from emp
where deptno = 20
order by 1,3
```

DEPTNO	ENAME	SAL
20	SMITH	800
20	ADAMS	1100
20	JONES	2975
20	SCOTT	3000
20	FORD	3000

要将所有的 SAL 字段值增加 10%。

### 解决方案

使用 UPDATE 语句来修改数据库表中已有行。例如：

```
1 update emp
2   set sal = sal*1.10
3   where deptno = 20
```

### 讨论

使用带有 WHERE 子句的 UPDATE 语句，可以指定哪些行需要更新。如果没有 WHERE 子句，则表中所有行都将被更新。在此解决方案中的表达式“SAL\*1.10”表示将工资增加 10%。

当准备要对大量的数据进行更新时，用户可能需要先预览一下结果，此时可以使用一个 SELECT 语句，该语句中包含想要放到 SET 子句中的表达式。下面的 SELECT 语句显示了工资增加 10% 后的结果：

```
select deptno,
       ename,
       sal      as orig_sal,
       sal*.10  as amt_to_add,
       sal*1.10 as new_sal
  from emp
 where deptno=20
 order by 1,5
```

DEPTNO	ENAME	ORIG_SAL	AMT_TO_ADD	NEW_SAL
20	SMITH	800	80	880
20	ADAMS	1100	110	1210
20	JONES	2975	298	3273
20	SCOTT	3000	300	3300
20	FORD	3000	300	3300

增加的工资被分为两列：一列是显示与原工资的差额，而另一列显示增加后的工资。

## 4.9 当相应行存在时更新

### 问题

仅当另一个表中相应的行存在时，更新某表中的一些行。例如，如果表 EMP\_BONUS 中存在某位员工，则要将该员工的工资增加 20%（在表 EMP 中）。下面的结果集列出了在表 EMP\_BONUS 中的当前数据：

```
select empno, ename
  from emp_bonus
```

EMPNO	ENAME
7369	SMITH
7900	JAMES
7934	MILLER

### 解决方案

为了可以将符合条件的员工工资增加 20%，可以在 UPDATE 语句的 WHERE 子句中使用子查询，用以找出哪些员工同时存在于表 EMP 和 EMP\_BONUS 中：

```
1 update emp
2   set sal=sal*1.20
3  where empno in ( select empno from emp_bonus )
```

### 讨论

子查询返回的结果集确定了在表 EMP 中哪些行可以被更新。谓词 IN 用来检验 EMP 中的 EMPNO 值是否包含在由子查询返回的 EMPNO 值列表中，当在此列表中时，相应的 SAL 值被更新。

还可以使用 EXISTS 子句来替代 IN:

```
update emp
  set sal = sal*1.20
  where exists ( select null
                  from emp_bonus
                  where emp.empno=emp_bonus.empno )
```

读者可能会奇怪在 EXISTS 子查询中 SELECT 列表中的 NULL 值, 不要惊讶, NULL 值对更新操作没有任何不利影响, 我认为这样既增加了该语句的可读性, 而且还强调了一个事实: 与使用 IN 操作符的子查询不同, 使用 EXISTS 的解决方案中, 由子查询的 WHERE 子句决定要更新哪些行, 而不是由子查询的 SELECT 列表中的值决定。

## 4.10 用其他表中的值更新

### 问题

要用一个表中的值来更新另外一个表中的行。例如, 在表 NEW\_SAL 中保存着某个特定员工的新工资, 表 NEW\_SAL 中内容如下:

```
select *
  from new_sal

DEPTNO      SAL
-----
10          4000
```

在表 NEW\_SAL 中, DEPTNO 为主关键字。要用表 NEW\_SAL 中的值更新表 EMP 中相应员工的工资, 条件是 EMP.DEPTNO 与 NEW\_SAL.DEPTNO 相等, 将匹配记录的 EMP.SAL 更新为 NEW\_SAL.SAL, 将 EMP.COMM 更新为 NEW\_SAL.SAL 的 50%。表 EMP 中的内容如下所示:

```
select deptno,ename,sal,comm
  from emp
 order by 1
```

DEPTNO	ENAME	SAL	COMM
10	CLARK	2450	
10	KING	5000	
10	MILLER	1300	
20	SMITH	800	
20	ADAMS	1100	
20	FORD	3000	
20	SCOTT	3000	
20	JONES	2975	
30	ALLEN	1600	300
30	BLAKE	2850	
30	MARTIN	1250	1400
30	JAMES	950	
30	TURNER	1500	0
30	WARD	1250	500

### 解决方案

在表 NEW\_SAL 和 EMP 之间做联接, 用来查找 COMM 新值并带给 UPDATE 语句。像这

样通过关联子查询进行更新的情况十分常见；另一种方法是创建视图（传统视图或内联视图，视数据库支持而定），然后更新这个视图。

## DB2 and MySQL

使用相关的子查询来设置表EMP中的SAL和COMM值，同样使用相关的子查询判别表EMP中哪些行需要被更新：

```
1 update emp e set (e.sal,e.comm) = (select ns.sal, ns.sal/2
2                                   from new_sal ns
3                                   where ns.deptno=e.deptno)
4 where exists ( select null
5               from new_sal ns
6               where ns.deptno = e.deptno )
```

## Oracle

DB2 解决方案的方法也适用，但是，Oracle 中还可以选择使用内联视图来进行更新：

```
1 update (
2   select e.sal as emp_sal, e.comm as emp_comm,
3          ns.sal as ns_sal, ns.sal/2 as ns_comm
4   from emp e, new_sal ns
5   where e.deptno = ns.deptno
6 ) set emp_sal = ns_sal, emp_comm = ns_comm
```

## PostgreSQL

DB2 解决方案的方法也适用，但 PostgreSQL 中还有另一种方法（更方便），就是在 UPDATE 语句中直接使用联接：

```
1 update emp
2   set sal = ns.sal,
3       comm = ns.sal/2
4   from new_sal ns
5   where ns.deptno = emp.deptno
```

## SQL Server

DB2 解决方案的方法也适用，但 SQL Server 还有一种方法（类似与 PostgreSQL 解决方案）就是在 UPDATE 语句中直接使用联接：

```
1 update e
2   set e.sal = ns.sal,
3       e.comm = ns.sal/2
4   from emp e,
5       new_sal ns
6   where ns.deptno = e.deptno
```

## 讨论

在讨论这些不同的解决方案之前，先要提几个有关使用查询进行更新的重要方面。关联子查询中的 WHERE 子句与更新语句的 WHERE 子句是不同的。看一下“问题”部分的 UPDATE 语句，在表 EMP 和 NEW\_SAL 之间按 DEPTNO 进行联接，其结果行返回到 UPDATE 语句的 SET 子句中。对于 DEPTNO 为 10 的员工，因为在表 NEW\_SAL 中存在

匹配的 DEPTNO, 故会返回有效值。但是, 对于其他部门的员工来说结果又是如何呢? NEW\_SAL 表中没有其他的部门, 所以对于在 DEPTNO 为 20 和 30 的员工, 他们的 SAL 和 COMM 将设为空。除非通过 LIMIT、TOP 或其他由供应商支持的机制来限制结果集中的数目, 在 SQL 中, 只有一种方法可以限制从表中返回的行数, 那就是使用 WHERE 子句。要正确的执行 UPDATE 语句, 对要更新的表使用 WHERE 子句, 同时, 在关联子查询中也使用 WHERE 子句。

## DB2 和 MySQL

如果不需要更新表 EMP 中的所有行, 记住一定要在 UPDATE 语句的 WHERE 子句中包含关联子查询。仅在 SET 子句中执行联接 (关联子查询) 是不够的。通过在 UPDATE 语句的 WHERE 子句, 可以确保在表 EMP 中, 只有 DEPTNO 与表 NEW\_SAL 匹配的行被更新。此法则通常对于所有的 RDBMS 都有效。

## Oracle

Oracle 的解决方案使用更新联接视图, 实际上是用等值联接来判别对那些行进行更新。单独执行一下查询就可以确定将更新哪些行。要成功地使用这种类型的 UPDATE 语句, 首先必须理解键值保留表 (key-preservation) 的概念。表 NEW\_SAL 的 DEPTNO 列是该表的主关键字, 这样, 它的值在该表中是唯一的。然而在表 EMP 和 NEW\_SAL 中进行联接时, NEW\_SAL.DEPTNO 在结果集中并不唯一, 则结果如下所示:

```
select e.empno, e.deptno e_dept, ns.sal, ns.deptno ns_deptno
from emp e, new_sal ns
where e.deptno = ns.deptno
```

EMPNO	E_DEPT	SAL	NS_DEPTNO
7782	10	4000	10
7839	10	4000	10
7934	10	4000	10

如果要使 Oracle 能够更新此联接, 联接中的一个表必须为键值保留表, 意思是如果它的值在结果集中不唯一, 至少这些值在来源表中是唯一的。本例中, 表 NEW\_SAL 的 DEPTNO 为关键字, 表示该字段值在表中是唯一的。因为它在该表中唯一, 所以它虽然在结果集中多次出现, 也认为是键值保留的, 这样, 更新就能成功地完成。

## PostgreSQL 和 SQL Server

这两种平台上的该语法稍有不同, 然而所使用的技巧是一样的。在 UPDATE 语句中直接做联接非常方便。因为指定了更新的目标表 (UPDATE 关键字后列出的表), 就不会混淆哪个表中的行要被修改。另外, 因为在更新中使用了联接 (因为有显式的 WHERE 子句), 可以避免关联子查询更新编码的错误; 还有, 如果在这里遗漏了联接, 显然表明此语句有问题。

## 4.11 合并记录

### 问题

根据表中记录存在状况，响应进行插入、更新或删除记录，条件为是否有相应的记录（如果记录存在，则更新；如果不存在，则插入，如果在更新之后不满足特定的条件，删除该记录）。例如，要按下列方式编辑表 EMP\_COMMISSION 中记录：

- 如果在表 EMP\_COMMISSION 中的某员工也存在于表 EMP 中，那么将他们的提成更新为 1000。
- 对于已经将其 COMM 值更新到 1000 的所有员工，如果他们的 SAL 值少于 2000，删除他们（这些记录在表 EMP\_COMMISSION 中将不存在）。
- 否则，从表 EMP 中取出 EMPNO、ENAME 和 DEPTNO 值，插入到表 EMP\_COMMISSION 中。

此问题的实质是想要根据表 EMP 中给出的行是否与表 EMP\_COMMISSION 中相应的行匹配，来执行 UPDATE 或 INSERT 操作。然后根据 UPDATE 的结果来删除那些提成太高的记录。

下面分别列出了在表 EMP 和表 EMP\_COMMISSION 中的行：

```
select deptno, empno, ename, comm
from emp
order by 1
```

DEPTNO	EMPNO	ENAME	COMM
10	7782	CLARK	
10	7839	KING	
10	7934	MILLER	
20	7369	SMITH	
20	7876	ADAMS	
20	7902	FORD	
20	7788	SCOTT	
20	7566	JONES	
30	7499	ALLEN	300
30	7698	BLAKE	
30	7654	MARTIN	1400
30	7900	JAMES	
30	7844	TURNER	0
30	7521	WARD	500

```
select deptno, empno, ename, comm
from emp_commission
order by 1
```

DEPTNO	EMPNO	ENAME	COMM
10	7782	CLARK	
10	7839	KING	
10	7934	MILLER	

## 解决方案

Oracle是目前唯一具有可以解决此问题的RDBMS。该语句为MERGE，它能够按需要执行UPDATE或INSERT操作，例如：

```
1 merge into emp_commission ec
2 using (select * from emp) emp
3   on (ec.empno=emp.empno)
4   when matched then
5     update set ec.comm = 1000
6     delete where (sal < 2000)
7   when not matched then
8     insert (ec.empno,ec.ename,ec.deptno,ec.comm)
9     values (emp.empno,emp.ename,emp.deptno,emp.comm)
```

## 讨论

在解决方案中第3行的联接用来确定那些行已经存在并将被更新。该联接是EMP\_COMMISSION（别名为EC）和子查询（别名为EMP）之间的联接。如果联接成功，则认为两行“匹配”，然后执行WHEN MATCHED子句中的UPDATE语句；否则，如果没有匹配的行，则执行在WHEN NOT MATCHED子句中的INSERT语句。因此，如果表EMP中的某些行，在表EMP\_COMMISSION中按DEPTNO字段没有对应的记录，则这些行将被插入到表EMP\_COMMISSION中。在表EMP的所有员工中，只有在部门10中的员工在表EMP\_COMMISSION中的COMM值被更新，而其他员工的信息将被插入到该表中；另外，员工MILLER虽然在DEPTNO 10中，相应的COMM值也被更新，但由于她的SAL值小于2000，所以他的记录从表EMP\_COMMISSION中删除。

## 4.12 从表中删除所有记录

### 问题

从表中删除所有记录。

## 解决方案

使用DELETE命令来删除表中的记录，例如，要删除表EMP中的所有记录：

```
delete from emp
```

## 讨论

当使用不带有WHERE子句的DELETE命令时，将会删除指定表中的所有记录。

## 4.13 删除指定记录

### 问题

从某个表中删除满足指定条件的记录。



## 解决方案

使用带有 WHERE 子句的 DELETE 命令，用来指定哪些行将被删除。例如，要删除部门 10 的员工：

```
delete from emp where deptno = 10
```

## 讨论

通过使用带有 WHERE 子句的 DELETE 命令，可以从表中删除部分行，而不是删除所有行。

### 4.14 删除单个记录

#### 问题

从表中删除单个记录。

## 解决方案

这是“删除指定记录”的一种特殊应用，关键在于要确认所选择条件很“狭窄”，可以确定要删除的记录，一般按关键字进行删除。例如，要删除员工 CLARK (EMPNO 为 7782)：

```
delete from emp where empno = 7782
```

## 讨论

删除操作都是要判别删除哪些行，而 DELETE 语句的作用范围总是由 WHERE 子句决定。如果省略了 WHERE 子句，则 DELETE 的作用范围是整个表。通过在 WHERE 子句中写入条件，可以控制删除的范围是一组记录或是单个记录。当删除单个记录时，一般使用主关键字或其他唯一的关键字来作为判别条件。

---

**警告：**如果删除条件是基于主关键字或唯一关键字的，那么可以确认只会删除一条记录（这是因为 RDBMS 将被不允许两条记录具有相同的主关键字或唯一关键字）。否则，在删除操作前要仔细检查，确认没有多删除记录。

---

### 4.15 删除违反参照完整性的记录

#### 问题

从表中删除哪些记录，它们是要引用其他表中不存在的记录。例如，某些员工被分配到了一个不存在的部门中，要将这些员工删除。

## 解决方案

用谓词 NOT EXISTS 和子查询的来判别部门号的是否合法：

```
delete from emp
where not exists (
  select * from dept
  where dept.deptno = emp.deptno
)
```

另外，也可以使用谓词 NOT IN 来书写此查询：

```
delete from emp
where deptno not in (select deptno from dept)
```

## 讨论

删除实际上就是选择：在 WHERE 子句中用条件正确表示所要删除记录。

NOT EXISTS 解决方案用关联子查询来判断 DEPT 中是否存在 DEPTNO 跟 EMP 中给定记录相匹配的记录，如果该记录存在，则保留表 EMP 中相应的记录，否则，删除相应的记录。用这种方法检查 EMP 中的每条记录。

IN 解决方案用子查询来产生一个合法部门列表，然后用该列表检查表 EMP 中每条记录的 DEPTNO，如果 EMP 中某记录的 DEPTNO 不在此列表中时，删除该记录。

## 4.16 删除重复记录

### 问题

从表中删除重复记录。考虑如下所示的表：

```
create table dupes (id integer, name varchar(10))

insert into dupes values (1, 'NAPOLEON')
insert into dupes values (2, 'DYNAMITE')
insert into dupes values (3, 'DYNAMITE')
insert into dupes values (4, 'SHE SELLS')
insert into dupes values (5, 'SEA SHELLS')
insert into dupes values (6, 'SEA SHELLS')
insert into dupes values (7, 'SEA SHELLS')

select * from dupes order by 1
```

ID	NAME
1	NAPOLEON
2	DYNAMITE
3	DYNAMITE
4	SHE SELLS
5	SEA SHELLS
6	SEA SHELLS
7	SEA SHELLS

对于每个像“SEA SHELLS”这样重复的姓名组，只任意保留其中的一个 ID，并将其余

的记录删除。例如，对“SEA SHELLS”这种情况，不需要关心是删除记录 5 和 6、6 和 7 还是 5 和 7，最终表中只有一条有关“SEA SHELLS”的记录。

## 解决方案

用带有聚集函数的子查询，例如 MIN，任意选择保留的 ID（本例中只保留每组中 ID 号最小的记录）：

```
1 delete from dupes
2   where id not in ( select min(id)
3                     from dupes
4                     group by name )
```

## 讨论

要删除重复记录，首先要明确定义两行互相“重复”的概念。在本节的例子中，两条记录“重复”的意思就是指 NAME 列的值相同。根据此定义，可以考察每个重复组中的其他列的差别，以识别需要保留的行，如果这些有差别的列（或几列）是主关键字是最理想的。本例可以通过 ID 列来识别，因为没有两条记录的 ID 号是相同的。

该解决方案的关键就在于，首先对重复的记录分组（本例通过 NAME 字段），然后使用聚集函数挑选出一个键值保留。解决方案中的子查询回返回每个 NAME 组的最小 ID 值，与其对应的行也就是想要保存的记录。

```
select min(id)
  from dupes
 group by name

MIN(ID)
-----
      2
      1
      5
      4
```

DELETE 语句随后删除在该表中没有被子查询返回所有记录（在这里，ID 为 3、6 和 7 的记录）。如果还是不能理解该语句的机理，可以首先单独运行该子查询，并将 NAME 放到 SELECT 列表中：

```
select name, min(id)
  from dupes
 group by name

NAME          MIN(ID)
-----
DYNAMITE             2
NAPOLEON             1
SEA SHELLS           5
SHE SELLS            4
```

由子查询返回的行被保留，在 DELETE 语句中的 NOT IN 谓词表示将所有其他的行全部删除。

## 4.17 删除从其他表引用的记录

### 问题

从一个表中删除被另一个表引用的记录。考虑下面的 DEPT\_ACCIDENTS 表，其中每行代表生产过程中的一次事故，每行中记录了事故发生的部门以及事故类型。

```
create table dept_accidents
( deptno      integer,
  accident_name varchar(20) )

insert into dept_accidents values (10,'BROKEN FOOT')
insert into dept_accidents values (10,'FLESH WOUND')
insert into dept_accidents values (20,'FIRE')
insert into dept_accidents values (20,'FIRE')
insert into dept_accidents values (20,'FLOOD')
insert into dept_accidents values (30,'BRUISED GLUTE')

select * from dept_accidents
```

DEPTNO	ACCIDENT_NAME
10	BROKEN FOOT
10	FLESH WOUND
20	FIRE
20	FIRE
20	FLOOD
30	BRUISED GLUTE

要从表 EMP 中删除所在部门已经发生了三次以上事故的所有员工的记录。

### 解决方案

使用子查询和聚集函数 COUNT 来查找出发生了三次事故以上的部门，然后将这些部门的员工全部删除：

```
1 delete from emp
2   where deptno in ( select deptno
3                     from dept_accidents
4                     group by deptno
5                     having count(*) >= 3 )
```

### 讨论

子查询用来检查那些发生过三次以上事故的部门：

```
select deptno
from dept_accidents
group by deptno
having count(*) >= 3
```

DEPTNO
20

然后，DELETE 命令将删除由子查询返回的部门中的所有员工（在本例中，只有部门20）。

# 元数据查询

本章介绍在给定模式中查找信息的有关内容，例如，哪些表已经创建，或者哪些外键没有被索引。在本书中介绍的所有 RDBMS 都有表或视图以提供这些数据。本章将给读者提供从这些表或视图中获取信息的方法。然而，实际的信息比这里介绍的要多得多，请参阅所使用的 RDBMS 的文档，查看所有的目录列表或数据字典表 / 视图。

注意：为了便于示范，本章均假定所用模式名为 SMEAGOL。

## 5.1 列出模式中的表

### 问题

查看在给定的模式中所有已创建的表的清单。

### 解决方案

该解决方案服从模式名为 SMEAGOL 的假设。解决方案中最基本的方案在所有 RDBMS 中都一样：查询一个包含着数据库中所有表名称的系统表（或视图）。

#### DB2

查询 SYSCAT.TABLES:

```
1 select tabname
2   from syscat.tables
3  where tabschema = 'SMEAGOL'
```

#### Oracle

查询 SYS.ALL\_TABLES:

```
select table_name
  from all_tables
```

```
where owner = 'SMEAGOL'
```

## PostgreSQL、MySQL 和 SQL Server

查询 INFORMATION\_SCHEMA.TABLES:

```
1 select table_name
2   from information_schema.tables
3  where table_schema = 'SMEAGOL'
```

## 讨论

值得高兴的是，数据库用来披露其自身信息的机制，跟我们应用程序中所使用的完全一样：表和视图。例如，Oracle 提供了一个大的系统视图编目，如 ALL\_TABLES，可以在其中查询有关表、索引、授权和其他数据库对象。

---

**注意：**Oracle 的编目视图就是一些视图，它们基于一些基表，基表中包含丰富的信息，但用户界面非常不好。这些视图将 Oracle 的编目数据用友好的用户界面表现出来。

---

Oracle 的系统视图和 DB2 的系统表互不相同；另一方面，PostgreSQL、MySQL 和 SQL Server 支持一种信息模式，该信息模式是 ISO SQL 标准定义的视图集，因此在这三个数据库中可以使用相同的查询。

## 5.2 列出表的列

### 问题

列出表的各列、它们的数据类型，以及这些列在表中的位置。

### 解决方案

在下面的解决方案中，假设想要列出在模式 SMEAGOL 中 EMP 表的各列、它们的数据类型及以数字表示的位置。

#### DB2

查询 SYSCAT.COLUMNS:

```
1 select colname, typename, colno
2   from syscat.columns
3  where tabname   = 'EMP'
4    and tabschema = 'SMEAGOL'
```

#### Oracle

查询 ALL\_TAB\_COLUMNS:

```
1 select column_name, data_type, column_id
2   from all_tab_columns
3  where owner       = 'SMEAGOL'
```

```
4      and table_name = 'EMP'
```

## PostgreSQL、MySQL 和 SQL Server

查询 INFORMATION\_SCHEMA.COLUMNS:

```
1 select column_name, data_type, ordinal_position
2   from information_schema.columns
3  where table_schema = 'SMEAGOL'
4    and table_name  = 'EMP'
```

## 讨论

每个版本的数据库都提供了获得列数据明细的方法。在此例子中，只返回了列名称、数据类型和位置，而其他常用的信息包括列长度、可否为空和默认值等项目。

## 5.3 列出表的索引列

### 问题

列出给定表的索引、索引的列及这些列在索引中的位置（如果可能）。

### 解决方案

对各种数据库都假定：想要列出 SMEAGOL 模式中表 EMP 的索引。

#### DB2

查询 SYSCAT.INDEXES:

```
1 select a.tabname, b.indname, b.colname, b.colseq
2   from syscat.indexes a,
3        syscat.indexcoluse b
3  where a.tabname   = 'EMP'
4        and a.tabschema = 'SMEAGOL'
5        and a.indschema = b.indschema
6        and a.indname  = b.indname
```

#### Oracle

查询 SYS.ALL\_IND\_COLUMNS:

```
select table_name, index_name, column_name, column_position
  from sys.all_ind_columns
 where table_name = 'EMP'
    and table_owner = 'SMEAGOL'
```

#### PostgreSQL

查询 PG\_CATALOG.PG\_INDEXES 和 INFORMATION\_SCHEMA.COLUMNS:

```
1 select a.tablename, a.indexname, b.column_name
2   from pg_catalog.pg_indexes a,
3        information_schema.columns b
4  where a.schemaname = 'SMEAGOL'
5    and a.tablename = b.table_name
```

## MySQL

使用 SHOW INDEX 命令：

```
show index from emp
```

## SQL Server

查询 SYS.TABLES、SYS.INDEXES、SYS.INDEX\_COLUMNS 和 SYS.COLUMNS：

```
1 select a.name table_name,
2        b.name index_name,
3        d.name column_name,
4        c.index_column_id
5   from sys.tables a,
6        sys.indexes b,
7        sys.index_columns c,
8        sys.columns d
9  where a.object_id = b.object_id
10     and b.object_id = c.object_id
11     and b.index_id = c.index_id
12     and c.object_id = d.object_id
13     and c.column_id = d.column_id
14     and a.name = 'EMP'
```

## 讨论

当开始查询时，需要知道哪些列是索引列，哪些列不是。对查询的筛选条件中经常用到或 fairly selective 的列，索引可以显著改进性能；当表之间进行联接时，索引也很有用。如果知道了哪些列是索引列，即使会出现性能问题，也已经在解决的路上先行一步了。另外，用户也可能想查找有关索引本身的信息：索引深度有多少层，有多少不相同的关键字、有多少叶子块等等。这些信息都可以从此解决方案所用的视图（或表）得到。

## 5.4 列出表约束

### 问题

列出某模式中对某表定义的约束以及这些约束所基于的列。例如，要查找表 EMP 中的约束及约束所基于的列。

### 解决方案

#### DB2

查询 SYSCAT.TABCONST 和 SYSCAT.COLUMNS：

```
1 select a.tabname, a.constname, b.colname, a.type
2   from syscat.tabconst a,
3        syscat.columns b
4  where a.tabname = 'EMP'
5     and a.tabschema = 'SMEAGOL'
6     and a.tabname = b.tabname
7     and a.tabschema = b.tabschema
```



## Oracle

查询 SYS.ALL\_CONSTRAINTS 和 SYS.ALL\_CONS\_COLUMNS:

```
1 select a.table_name,
2       a.constraint_name,
3       b.column_name,
4       a.constraint_type
5   from all_constraints a,
6       all_cons_columns b
7  where a.table_name    = 'EMP'
8        and a.owner      = 'SMEAGOL'
9        and a.table_name = b.table_name
10       and a.owner       = b.owner
11       and a.constraint_name = b.constraint_name
```

## PostgreSQL、MySQL 和 SQL Server

查询 INFORMATION\_SCHEMA.TABLE\_CONSTRAINTS 和 INFORMATION\_SCHEMA.KEY\_COLUMN\_USAGE:

```
1 select a.table_name,
2       a.constraint_name,
3       b.column_name,
4       a.constraint_type
5   from information_schema.table_constraints a,
6       information_schema.key_column_usage b
7  where a.table_name    = 'EMP'
8        and a.table_schema = 'SMEAGOL'
9        and a.table_name = b.table_name
10       and a.table_schema = b.table_schema
11       and a.constraint_name = b.constraint_name
```

## 讨论

约束是关系数据库的关键部分,要知道在表中有什么样的约束是理所当然的要求。列出在表上的约束有很多方面的用处:可能要查找缺少关键字的表、可能要查找出那些本来应当是但并不是外键的列(也就是说,子表的数据与父表数据不同,要知道这是什么原因造成的),或者想要知道有关检查约束的内容(列是否能够为空?是否需要满足特定的条件?等等)。

## 5.5 列出没有相应索引的外键

### 问题


列出含有没有被索引的外键的表。例如,判断表 EMP 中外键是否被索引。

### 解决方案

#### DB2

查询 SYSCAT.TABCONST、SYSCAT.KEYCOLUSE、SYSCAT.INDEXES 和 SYSCAT.INDEXCOLUSE:

```
1 select fkeys.tabname,
2       fkeys.constname,
3       fkeys.colname,
4       ind_cols.indname
```



```

5   from (
6   select a.tabschema, a.tabname, a.constname, b.colname
7   from syscat.tabconst a,
8   syscat.keycoluse b
9   where a.tabname = 'EMP'
10    and a.tabschema = 'SMEAGOL'
11    and a.type = 'F'
12    and a.tabname = b.tabname
13    and a.tabschema = b.tabschema
14   ) fkeys
15   left join
16   (
17   select a.tabschema,
18   a.tabname,
19   a.indname,
20   b.colname
21   from syscat.indexes a,
22   syscat.indexcoluse b
23   where a.indschema = b.indschema
24   and a.indname = b.indname
25   ) ind_cols
26   on ( fkeys.tabschema = ind_cols.tabschema
27   and fkeys.tabname = ind_cols.tabname
28   and fkeys.colname = ind_cols.colname )
29   where ind_cols.indname is null

```

## Oracle

查询 SYS.ALL\_CONS\_COLUMNS、SYS.ALL\_CONSTRAINTS 和 SYS.ALL\_IND\_COLUMNS:

```

1   select a.table_name,
2   a.constraint_name,
3   a.column_name,
4   c.index_name
5   from all_cons_columns a,
6   all_constraints b,
7   all_ind_columns c
8   where a.table_name = 'EMP'
9   and a.owner = 'SMEAGOL'
10  and b.constraint_type = 'R'
11  and a.owner = b.owner
12  and a.table_name = b.table_name
13  and a.constraint_name = b.constraint_name
14  and a.owner = c.table_owner (+)
15  and a.table_name = c.table_name (+)
16  and a.column_name = c.column_name (+)
17  and c.index_name is null

```

## PostgreSQL

查询 INFORMATION\_SCHEMA.KEY\_COLUMN\_USAGE、INFORMATION\_SCHEMA.REFERENTIAL\_CONSTRAINTS、INFORMATION\_SCHEMA.COLUMNS 和 PG\_CATALOG.PG\_INDEXES:

```

1   select fkeys.table_name,
2   fkeys.constraint_name,
3   fkeys.column_name,
4   ind_cols.indexname
5   from (
6   select a.constraint_schema,
7   a.table_name,

```

```

8         a.constraint_name,
9         a.column_name
10    from information_schema.key_column_usage a,
11         information_schema.referential_constraints b
12   where a.constraint_name = b.constraint_name
13         and a.constraint_schema = b.constraint_schema
14         and a.constraint_schema = 'SMEAGOL'
15         and a.table_name = 'EMP'
16         ) fkeys
17   left join
18   (
19   select a.schemaname, a.tablename, a.indexname, b.column_name
20     from pg_catalog.pg_indexes a,
21          information_schema.columns b
22   where a.tablename = b.table_name
23         and a.schemaname = b.table_schema
24         ) ind_cols
25   on ( fkeys.constraint_schema = ind_cols.schemaname
26        and fkeys.table_name = ind_cols.tablename
27        and fkeys.column_name = ind_cols.column_name )
28   where ind_cols.indexname is null

```

## MySQL

可以使用 SHOW INDEX 命令来检索索引信息，如索引名、在索引中的列和在索引中这些列的顺序。另外，可以查询 INFORMATION\_SCHEMA.KEY\_COLUMN\_USAGE 列出指定表的外键。在 MySQL 5 中，外键号称可以自动索引，但事实上是可以解除索引的。要检测外键索引是否被解除，可以对该表执行 SHOW INDEX 命令，并且将结果跟 INFORMATION\_SCHEMA.KEY\_COLUMN\_USAGE.COLUMN\_NAME 中同一表的结果比较，如果 KEY\_COLUMN\_USAGE 中有 COLUMN\_NAME，但是 SHOW INDEX 的结果中没有，则该列没有被索引。

## SQL Server

查询 SYS.TABLES、SYS.FOREIGN\_KEYS、SYS.COLUMNS、SYS.INDEXES 和 SYS.INDEX\_COLUMNS:

```

1  select fkeys.table_name,
2         fkeys.constraint_name,
3         fkeys.column_name,
4         ind_cols.index_name
5    from (
6  select a.object_id,
7         d.column_id,
8         a.name table_name,
9         b.name constraint_name,
10        d.name column_name
11    from sys.tables a
12   join
13   sys.foreign_keys b
14  on ( a.name = 'EMP'
15      and a.object_id = b.parent_object_id
16      )
17   join
18   sys.foreign_key_columns c
19  on ( b.object_id = c.constraint_object_id )
20   join
21   sys.columns d
22  on ( c.constraint_column_id = d.column_id
23      and a.object_id = d.object_id
24      )

```

```

25         ) fkeys
26     left join
27     (
28 select a.name index_name,
29        b.object_id,
30        b.column_id
31     from sys.indexes a,
32          sys.index_columns b
33    where a.index_id = b.index_id
34        ) ind_cols
35     on (      fkeys.object_id = ind_cols.object_id
36            and fkeys.column_id = ind_cols.column_id )
37    where ind_cols.index_name is null

```

## 讨论

在修改行时，每种数据库都使用其自己的加锁机制，如果存在由外键建立的父-子关联，在子列上建索引可以减少锁定（具体细节查看所使用的RDBMS文档）；其他情况下，子表通常按外键跟父表联接，这种情况下索引也有助于提高性能。

## 5.6 使用 SQL 来生成 SQL

### 问题

创建动态 SQL 语句，使之自动完成维护任务。例如，要完成如下的 3 个任务：计算所有表中的行数、禁用所有表中定义的外键约束及用表中数据中生成插入操作脚本。

### 解决方案

这种解决方案就是使用字符串来创建 SQL 语句，并且 SQL 语句中需要填进的值（如命令要执行的对象名称）将从所要选择的表中获得。注意，此查询只生成语句，用户通常必须通过脚本、手工或其他执行 SQL 语句的方法来执行这些语句。下面列出的例子是 Oracle 系统下的查询，对于其他的 RDBMS，这种技术也是相同的，唯一不同的地方就是诸如数据字典表的名称和日期格式等。在下面列出的查询输出内容是在笔者电脑的 Oracle 实例所返回行中的一部分，读者的结果集将与之有所不同。

```

/* 生成 SQL 来计算所有表中的行数 */
select 'select count(*) from '||table_name||';' cnts
  from user_tables;

CNTS
-----
select count(*) from ANT;
select count(*) from BONUS;
select count(*) from DEMO1;
select count(*) from DEMO2;
select count(*) from DEPT;
select count(*) from DUMMY;
select count(*) from EMP;
select count(*) from EMP_SALES;
select count(*) from EMP_SCORE;
select count(*) from PROFESSOR;
select count(*) from T;
select count(*) from T1;
select count(*) from T2;

```

```

select count(*) from T3;
select count(*) from TEACH;
select count(*) from TEST;
select count(*) from TRX_LOG;
select count(*) from X;

/* 禁用所有表中的外键 */
select 'alter table '||table_name||
       ' disable constraint '||constraint_name||',' cons
from user_constraints
where constraint_type = 'R';

CONS
-----
alter table ANT disable constraint ANT_FK;
alter table BONUS disable constraint BONUS_FK;
alter table DEMO1 disable constraint DEMO1_FK;
alter table DEMO2 disable constraint DEMO2_FK;
alter table DEPT disable constraint DEPT_FK;
alter table DUMMY disable constraint DUMMY_FK;
alter table EMP disable constraint EMP_FK;
alter table EMP_SALES disable constraint EMP_SALES_FK;
alter table EMP_SCORE disable constraint EMP_SCORE_FK;
alter table PROFESSOR disable constraint PROFESSOR_FK;

/* 根据表 EMP 中的某些列生成插入脚本 */
select 'insert into emp(empno,ename,hiredate) '||chr(10)||
       'values( '||empno||','||' '||ename
       ||','||to_date('||' '||hiredate||' '||') );' inserts
from emp
where deptno = 10;

INSERTS
-----
insert into emp(empno,ename,hiredate)
values( 7782,'CLARK',to_date('09-JUN-1981 00:00:00') );
insert into emp(empno,ename,hiredate)
values( 7839,'KING',to_date('17-NOV-1981 00:00:00') );
insert into emp(empno,ename,hiredate)
values( 7934,'MILLER',to_date('23-JAN-1982 00:00:00') );

```

## 讨论

使用SQL来生成SQL语句在创建可移植脚本方面非常有用，例如可能需要在多种环境中测试。另外，正如上面的例子中所见，使用SQL来生成SQL对于执行批量维护也是十分方便，并且还可以一次查询出多个对象的信息。使用SQL来生成SQL是非常简单的操作，如果对其多次练习的话还会变得更容易。给出的例子为读者生成自己的“动态”SQL脚本提供了很好的基础，因为说实话，有关这方面也就这么多了。多练习就会掌握。

## 5.7 在 Oracle 中描述数据字典视图

### 问题

使用的数据库平台为 Oracle，但无法记住可利用的数据字典视图，也记不住列定义，更糟的是，连 Oracle 的说明文档也找不到了。

## 解决方案

此节内容是专门针对 Oracle 数据库的。Oracle 不仅维护了一系列鲁棒数据字典视图，它还有用来说明这些数据字典视图的数据字典视图，真是一个精彩的闭环。

查询名为 DICTIONARY 的视图，用来列出数据字典视图和它们的用途：

```
select table_name, comments
       from dictionary
       order by table_name;
```

TABLE_NAME	COMMENTS
ALL_ALL_TABLES	Description of all object and relational tables accessible to the user
ALL_APPLY	Details about each apply process that dequeues from the queue visible to the 当前 user
...	

查询 DICT\_COLUMNS 来描述给出的数据字典视图中的列：

```
select column_name, comments
       from dict_columns
       where table_name = 'ALL_TAB_COLUMNS';
```

COLUMN_NAME	COMMENTS
OWNER	
TABLE_NAME	Table, view or cluster name
COLUMN_NAME	Column name
DATA_TYPE	Datatype of the column
DATA_TYPE_MOD	Datatype modifier of the column
DATA_TYPE_OWNER	Owner of the datatype of the column
DATA_LENGTH	Length of the column in bytes
DATA_PRECISION	Length: decimal digits (NUMBER) or binary digits (FLOAT)

## 讨论

以前，Oracle 的文档集还不能像现在这样在 Web 上轻易地检索到，但是使用 DICTIONARY 和 DICT\_COLUMNS 视图非常方便。只需知道这两个视图，就可以引导用户了解其他所有的视图进而了解整个的数据库。

就是现在，了解有关 DICTIONARY 和 DICT\_COLUMNS 这两个视图的知识也会带来很大方便。如果不能确认哪个视图描述给定的对象类型，可以使用带有通配符的查询来查找出来。例如，想要找到模式中哪些视图可能会对表做描述的线索：

```
select table_name, comments
       from dictionary
       where table_name LIKE '%TABLE%'
       order by table_name;
```

该查询返回所有包含字符串“TABLE”的数据字典视图，该方法利用了 Oracle 的数据字典视图命名规范相当一致这一特点，描述表的视图，其名称中大多包含“TABLE”（有些情况下，在 ALL\_TAB\_COLUMNS 中会将“TABLE”缩写为“TAB”）。

# 使用字符串

本章着重讨论在 SQL 语句中字符串的操作。要记住，SQL 的目标不是完成复杂的字符串操作，并且用户也会（也将）发现在 SQL 中对字符串进行操作是非常繁琐和令人厌烦的。SQL 虽然有所局限，各种 DBMS 都提供了一些非常有用的内置函数，笔者会以一些独创性的方法使用这些函数。本章尤其能够说明笔者引言部分要想表达的信息，SQL 既好、又不好，有时甚至很丑陋。笔者希望，通过本章，用户可以更好地判别，在 SQL 中对字符串进行操作时，哪些是可行的，哪些是不可行的。在很多情况下，读者可能会惊讶在 SQL 中对字符串的分解及转换是如此的容易，而有些时候，也可能被 SQL 必须要完成的特殊任务所震惊。

本章第 6.1 节非常重要，后来很多其他解决方案都要用到它。在很多情况下需要通过每次移动一个字符的方法来遍历字符串，但遗憾的是，使用 SQL 就没有这样简单了，因为在 SQL 中没有循环功能（Oracle 的 MODEL 子句除外），这就需要模拟一个循环来遍历字符串。笔者将这种操作叫做“走过字符串”或者“穿越字符串”，并且一开始就解释了这种技术。当使用 SQL 时，这是在字符串分析中的基础操作，在本章中所有解决方案中几乎都引用或使用这种操作。笔者强烈建议吃透这种技术的机理。

## 6.1 遍历字符串

### 问题

遍历一个字符，并将其中的每个字符都作为一行返回，但是 SQL 没有循环操作。例如，要将表 EMP 中 ENAME 值为“KING”的字符串显示为 4 行，每行中都包含“KING”中的一个字符。

### 解决方案

使用笛卡儿积来生成行号，用来在该行中返回字符串中的每个字符。然后，使用 DBMS

中的内置的字符串分析函数来摘出所要显示的字符（SQL Server 用户可使用 SUBSTRING 代替 SUBSTR）：

```
1 select substr(e.ename,iter.pos,1) as C
2   from (select ename from emp where ename = 'KING') e,
3        (select id as pos from t10) iter
4  where iter.pos <= length(e.ename)

C
-
K
I
N
G
```

## 讨论

逐一访问字符串中字符的关键是，所联接的表要有足够的行来得到所需要的反复次数。例中所用的表为 T10，该表包含 10 行（该表有 ID 列，其中存储了数字 1\_10）。示例查询中能够返回的行数最大为 10。

下面的例子显示了没有分解 ENAME 时在 E 和 ITER（也就是说，指定的姓名和 T10 中这 10 行）的笛卡儿积：

```
select ename, iter.pos
  from (select ename from emp where ename = 'KING') e,
       (select id as pos from t10) iter
```

ENAME	POS
KING	1
KING	2
KING	3
KING	4
KING	5
KING	6
KING	7
KING	8
KING	9
KING	10

内联视图的基数为 1，而内联视图 ITER 的基数为 10，所以笛卡儿积为 10 行。生成这样的结果是在 SQL 中进行循环的第一步。

---

注意：一般情况下，将表 T10 称为“基于 1”表。

---

解决方案使用 WHERE 子句在返回 4 行之后，跳出循环。为限定结果集的行数与在姓名中所包含的字符数相同，所以在其中定义 WHERE 子句用了 ITER.POS <= LENGTH(E.ENAME) 作为条件：

```
select ename, iter.pos
  from (select ename from emp where ename = 'KING') e,
       (select id as pos from t10) iter
 where iter.pos <= length(e.ename)
```



ENAME	POS
KING	1
KING	2
KING	3
KING	4

现在, 对于在E.ENAME中的每个字符都有对应的一行, 可以使用ER.POS作为SUBSTR的参数, 在字符串中操作这些字符。ITER.PRS 每行都在增加, 这样, 每行都会返回E.ENAME中的下一个字符。这也就是示例解决方案的工作原理。

根据所要实现的目标, 可以决定是否需要对在字符串中每个字符都生成单独的一行。下面的例子是要遍历E.ENAME, 并且显示字符串中的各个部分 (超过1个字符):

```
select substr(e.ename,iter.pos) a,
       substr(e.ename,length(e.ename)-iter.pos+1) b
  from (select ename from emp where ename = 'KING') e,
       (select id pos from t10) iter
 where iter.pos <= length(e.ename)
```

A	B
KING	G
ING	NG
NG	ING
G	KING

本章中的多数解决方案都有共同的部分, 例如将整个字符串中的每个字符拆分为一行, 或者根据字符串中特殊字符或分隔符的数量产生相应数量的行。

## 6.2 字符串文字中包含引号

### 问题

如要用 SQL 语句得到类似下面的结果, 字符串文字中要用到引号:

```
QMARKS
-----
g'day mate
beavers' teeth
'
```

### 解决方案

下面的3个SELECT代表了表示引号的3种方法: 嵌在字符串中间或单独表示:

```
1 select 'g'day mate' qmarks from t1 union all
2 select 'beavers'' teeth'   from t1 union all
3 select ''''                from t1
```

### 讨论

在使用引号时, 可以将它们当括号看待。如果有一个前括号, 也必须有一个相应的后括号, 这一原则也适用于引号。注意, 在任何字符串中, 必须保持引号个数为偶数。即若要在字符串中嵌入一个引号, 必然会有第二个引号:

```

select 'apples core', 'apple's core',
       case when '' is null then 0 else 1 end
from t1

'APPLESCORE' 'APPLE'SCOR' CASEWHEN''ISNULLTHEN0ELSE1END
-----
apples core  apple's core                                0

```

下面的解决方案一直从外向里剥，直到主要元素。最外层有使用两个引号定义了一个字符串，而在文字内部有两个挨在一起的引号，它们表示最终字符串中只得到一个引号。

```

Select ''' as quote from t1
Q
-
,

```

在使用引号时，一定要记住字符串是由两个引号来定义的，而在两个引号中没有任何字符时，表示 NULL 值。

## 6.3 计算字符在字符串中出现的次数

### 问题

计算一个字符或子串在给定的字符串中出现的次数。考虑下面的字符串：

```
10,CLARK,MANAGER
```

要计算在这个字符串中有多少个逗号。

### 解决方案

首先计算出原字符串的长度，然后计算去掉逗号后字符串的长度，这两者的差就是逗号在该字符串中出现的次数。每种DBMS都提供了求字符串长度以及从字符串中删除某个字符的函数。在多数情况下，这两个函数分别为LENGTH和REPLACE（SQL Server用户可以使用内置的LEN函数，而不是LENGTH）。

```

1 select (length('10,CLARK,MANAGER')-
2         length(replace('10,CLARK,MANAGER',',',''))) / length(',')
3         as cnt
4   from t1

```

### 讨论

这种解决方案只是使用一下减法。在第一行中调用了LENGTH函数来返回字符串的原始长度，然后在第二行中首次调用LENGTH函数是为了求得去掉逗号后的字符串长度，去逗号的操作用了REPLACE函数。

通过计算这两个长度的差可以得到这两个字符串中字符数的差，也就是在这个字符串中有多少个逗号。最后的一步操作就是将计算出的差值除以要查找的字符串的长度。如果所要查找的字符串的长度大于1时，此时这一步操作是必须的。在下面的示例中，需要

计算在字符串“HELLO HELLO”中字符串“LL”出现的字数，此时如果不使用除法将返回错误的结果。

```
select
  (length('HELLO HELLO')-
   length(replace('HELLO HELLO','LL','')))/length('LL')
  as correct_cnt,
  (length('HELLO HELLO')-
   length(replace('HELLO HELLO','LL',''))) as incorrect_cnt
from t1
```

CORRECT_CNT	INCORRECT_CNT
2	4

## 6.4 从字符串中删除不需要的字符问题

从数据中删除指定的字符，考虑下列的结果集：

ENAME	SAL
SMITH	800
ALLEN	1600
WARD	1250
JONES	2975
MARTIN	1250
BLAKE	2850
CLARK	2450
SCOTT	3000
KING	5000
TURNER	1500
ADAMS	1100
JAMES	950
FORD	3000
MILLER	1300

要得到如下所示的 STRIPPED1 和 STRIPPED2 列中的值，删除所有的 0 和元音字母：

ENAME	STRIPPED1	SAL	STRIPPED2
SMITH	SMTH	800	8
ALLEN	LLN	1600	16
WARD	WRD	1250	125
JONES	JNS	2975	2975
MARTIN	MRTN	1250	125
BLAKE	BLK	2850	285
CLARK	CLRK	2450	245
SCOTT	SCTT	3000	3
KING	KNG	5000	5
TURNER	TRNR	1500	15
ADAMS	DMS	1100	11
JAMES	JMS	950	95
FORD	FRD	3000	3
MILLER	MLLR	1300	13

## 解决方案

每个 DBMS 都提供了函数用来从字符串中删除不需要的字符，对于本问题来说，常用的就是 REPLACE 和 TRANSLATE 函数。

## DB2

使用内置的 TRANSLATE 和 REPLACE 函数来删除不需要的字符或字符串：

```
1 select  ename,
2         replace(translate(ename,'aaaaa','AEIOU'),'a','') stripped1,
3         sal,
4         replace(cast(sal as char(4)),'0','') stripped2
5   from emp
```

## MySQL 和 SQL Server

MySQL 和 SQL Server 不支持 TRANSLATE 函数，所以就需要多次调用 REPLACE 函数来执行此操作：

```
1 select  ename,
2         replace(
3         replace(
4         replace(
5         replace(
6         replace(ename,'A',''),'E',''),'I',''),'O',''),'U','')
7         as stripped1,
8         sal,
9         replace(sal,0,'') stripped2
10    from emp
```

## Oracle 和 PostgreSQL

使用内置的 TRANSLATE 和 REPLACE 函数来删除不需要的字符或字符串：

```
1 select  ename,
2         replace(translate(ename,'AEIOU','aaaaa'),'a')
3         as stripped1,
4         sal,
5         replace(sal,0,'') as stripped2
6   from emp
```

## 讨论

使用内置的 REPLACE 函数删除字符串中所有的 0，而要删除元音字母，使用 TRANSLATE 将所有的元音字符转换为一个指定的字符（在这里使用“a”，可以使用任意字符），然后使用 REPLACE 函数将该字符从字符串中全部删除掉。

## 6.5 将字符和数字数据分离

### 问题

在一列中数字数据和字符数据混合存储在一起（真不幸），要将这些数据中的数字和字符分离出来。考虑下面列出的数据集：

```
DATA
-----
SMITH800
ALLEN1600
WARD1250
JONES2975
MARTIN1250
```

```

BLAKE2850
CLARK2450
SCOTT3000
KING5000
TURNER1500
ADAMS1100
JAMES950
FORD3000
MILLER1300

```

而想要的结果如下列所示：

ENAME	SAL
SMITH	800
ALLEN	1600
WARD	1250
JONES	2975
MARTIN	1250
BLAKE	2850
CLARK	2450
SCOTT	3000
KING	5000
TURNER	1500
ADAMS	1100
JAMES	950
FORD	3000
MILLER	1300

## 解决方案

使用内置函数TRANSLATE和REPLACE将这些数字数据与字符数据分离，与本章中其他节所使用的方法类似，关键就是使用TRANSLATE函数来将多个字符转换为单个字符。这样就不需要查找许多数字或字符，而只是用一个字符来代表所有的数字或一个字符代表所有的字符。

### DB2

使用函数TRANSLATE和REPLACE来将数字与字符分离。

```

1 select replace(
2     translate(data,'0000000000','0123456789'),'0','') ename,
3     cast(
4     replace(
5     translate(lower(data),repeat('z',26),
6     'abcdefghijklmnopqrstuvwxyz'),'z','') as integer) sal
7 from (
8 select ename||cast(sal as char(4)) data
9 from emp
10 ) x

```

### Oracle

使用函数TRANSLATE和REPLACE来将数字与字符分离。

```

1 select replace(
2     translate(data,'0123456789','0000000000'),'0') ename,
3     to_number(
4     replace(
5     translate(lower(data),
6     'abcdefghijklmnopqrstuvwxyz',
7

```

```

8          rpad('z',26,'z')),'z')) sal
9      from (
10 select ename||sal data
11      from emp
12      )

```

## PostgreSQL

使用函数 TRANSLATE 和 REPLACE 来将数字与字符分离。

```

1 select replace(
2     translate(data,'0123456789','0000000000'),'0','') as ename,
3     cast(
4     replace(
5     translate(lower(data),
6     'abcdefghijklmnopqrstuvwxyz',
7     rpad('z',26,'z')),'z','') as integer) as sal
8  from (
9  select ename||sal as data
10  from emp
11  ) x

```

## 讨论

虽然对于每个 DBMS 来说，所使用的语法都有细微的差别，但是技术原理是一样的。现在用 Oracle 的解决方案来进行讨论。解决这个问题的关键就是将字符和数字隔离开。可以使用 TRANSLATE 和 REPLACE 函数来进行此项操作。要提取出数字数据，首先要使用 TRANSLATE 函数将所有的字符数据隔开。

```

select data,
       translate(lower(data),
                 'abcdefghijklmnopqrstuvwxyz',
                 rpad('z',26,'z')) sal
  from (select ename||sal data from emp)

```

DATA	SAL
SMITH800	zzzzz800
ALLEN1600	zzzzz1600
WARD1250	zzzzz1250
JONES2975	zzzzz2975
MARTIN1250	zzzzz1250
BLAKE2850	zzzzz2850
CLARK2450	zzzzz2450
SCOTT3000	zzzzz3000
KING5000	zzzzz5000
TURNER1500	zzzzz1500
ADAMS1100	zzzzz1100
JAMES950	zzzzz950
FORD3000	zzzzz3000
MILLER1300	zzzzz1300

通过使用 TRANSLATE 函数，首先将每个不是数字的字符转换为小写字母 Z，下一步是使用 REPLACE 函数将每条记录中的小写字母 Z 删除，只剩下可以转换为数值的数字字符。

```

select data,
       to_number(
         replace(
           translate(lower(data),
                     'abcdefghijklmnopqrstuvwxyz',

```

```

      rpad('z',26,'z')),'z')) sal
from (select ename||sal data from emp)

DATA                                SAL
-----
SMITH800                            800
ALLEN1600                           1600
WARD1250                             1250
JONES2975                            2975
MARTIN1250                           1250
BLAKE2850                            2850
CLARK2450                             2450
SCOTT3000                            3000
KING5000                             5000
TURNER1500                           1500
ADAMS1100                             1100
JAMES950                              950
FORD3000                             3000
MILLER1300                           1300

```

要提取出非数字字符，首先使用 TRANSLATE 函数隔离数字字符。

```

select data,
       translate(data,'0123456789','0000000000') ename
from (select ename||sal data from emp)

DATA                                ENAME
-----
SMITH800                            SMITH000
ALLEN1600                           ALLEN0000
WARD1250                             WARD0000
JONES2975                            JONES0000
MARTIN1250                           MARTIN0000
BLAKE2850                            BLAKE0000
CLARK2450                             CLARK0000
SCOTT3000                            SCOTT0000
KING5000                             KING0000
TURNER1500                           TURNER0000
ADAMS1100                             ADAMS0000
JAMES950                              JAMES000
FORD3000                             FORD0000
MILLER1300                           MILLER0000

```

通过使用 TRANSLATE 函数，可以将每个数字字符转换为零，下一步是使用 REPLACE 函数删除每条记录中所有的字符 0，结果只剩下非数字字符。

```

select data,
       replace(translate(data,'0123456789','0000000000'),'0') ename
from (select ename||sal data from emp)

DATA                                ENAME
-----
SMITH800                            SMITH
ALLEN1600                           ALLEN
WARD1250                             WARD
JONES2975                            JONES
MARTIN1250                           MARTIN
BLAKE2850                            BLAKE
CLARK2450                             CLARK
SCOTT3000                            SCOTT
KING5000                             KING
TURNER1500                           TURNER
ADAMS1100                             ADAMS
JAMES950                              JAMES
FORD3000                             FORD
MILLER1300                           MILLER

```

将这两种方法合并，就是本节所需要的解决方案。

## 6.6 判别字符串是不是字母数字型的问题

从表中返回某列只包含数字或字母，而不含任何其他字符的行。考虑下面列出的视图 V (SQL Server 用户使用操作符 “+” 来代替串联操作符 “||”)。

```
create view V as
select ename as data
  from emp
 where deptno=10
 union all
select ename||', $'|| cast(sal as char(4)) ||'.00' as data
  from emp
 where deptno=20
 union all
select ename|| cast(deptno as char(4)) as data
  from emp
 where deptno=30
```

视图 V 代表要操作的表，其内容如下所示：

```
DATA
-----
CLARK
KING
MILLER
SMITH, $800.00
JONES, $2975.00
SCOTT, $3000.00
ADAMS, $1100.00
FORD, $3000.00
ALLEN30
WARD30
MARTIN30
BLAKE30
TURNER30
JAMES30
```

然而，要从这个视图的数据中仅返回如下的记录：

```
DATA
-----
CLARK
KING
MILLER
ALLEN30
WARD30
MARTIN30
BLAKE30
TURNER30
JAMES30
```

简言之，就是想要去掉那些包含有非字母数字数据的记录。

### 解决方案

要解决此问题，一般人第一感觉就应该是：在字符串中查找所有可能出现的非数字字母



字符，但是恰恰相反，反过来做要容易得多：查出所有的字母与数字字符，这样就可以将它们转换为同一个字符，将所有的字母数字作为一个字符处理。这样做的原因是可以把这些字母与数字的字符作为一个整体来进行操作。一旦将所有字母与数字用所选定的字符替代之后，就可以很轻松地将字母数字与其他字符分开。

## DB2

使用TRANSLATE函数将所有的字母与数字字符转换为同一个字符，然后判别哪些行中还有没有被换转的字符。对于DB2的用户，必须在视图V中调用CAST函数，否则会由于类型转换错误而不能创建视图。在转换为CHAR类型时要特别注意，因为该类型的数据是定长的（会自动补空）。

```
1 select data
2   from V
3  where translate(lower(data),
4                  repeat('a',36),
5                  '0123456789abcdefghijklmnopqrstuvwxyz') =
6                  repeat('a',length(data))
```

## MySQL

在MySQL中，创建视图V的语法略有不同：

```
create view V as
select ename as data
  from emp
 where deptno=10
 union all
select concat(ename,', $',sal, '.00') as data
  from emp
 where deptno=20
 union all
select concat(ename,deptno) as data
  from emp
 where deptno=30
```

使用正则表达式可以很容易地找到那些包含非数字字母的数据：

```
1 select data
2   from V
3  where data regexp '[^0-9a-zA-Z]' = 0
```

## Oracle 和 PostgreSQL

使用TRANSLATE函数将所有的字母与数字字符转换为同一个字符，然后判别哪些行中有没有被换转的字符，对Oracle和PostgreSQL，不需要在视图V中调用CAST函数。在转换为CHAR类型时要特别注意，因为该类型的数据是定长的（会自动补空）。如果要进行转换，则转换为VARCHAR或VARCHAR2类型。

```
1 select data
2   from V
3  where translate(lower(data),
4                  '0123456789abcdefghijklmnopqrstuvwxyz',
5                  rpad('a',36,'a')) = rpad('a',length(data),'a')
```

## SQL Server

SQL Server 不支持 TRANSLATE 函数，因此必须判断每一行中只包含非数字字母值。有很多方法可以实现，下面给出的解决方案是使用了求 ASCII 值的方法。

```

1  select data
2    from (
3  select v.data, iter.pos,
4         substring(v.data,iter.pos,1) c,
5         ascii(substring(v.data,iter.pos,1)) val
6    from v,
7         ( select id as pos from t100 ) iter
8   where iter.pos <= len(v.data)
9         ) x
10   group by data
11   having min(val) between 48 and 122

```

## 讨论

此解决方案的关键就是能够同时代表多个字符。使用 TRANSLATE 函数，可以很轻松地操作所有的数字或所有的字母，而不需要挨个对字符进行检查。

## DB2、Oracle 和 PostgreSQL

在视图 V 中，14 行中只有 9 行是由字母和数字组成的，要查找出这些字母和数字所在的行，只需使用函数 TRANSLATE。在本示例中，TRANSLATE 函数将数字 0~9 和字母 a~z 全部转换为字母 a。一旦转换完成，被转换的行将与内容全是字母“a”的等长字符串进行比较，如果相同，则此行就全部是由字母和数字组成，没有其他的字符。

通过使用 TRANSLATE 函数（使用 Oracle 语法）：

```

where translate(lower(data),
               '0123456789abcdefghijklmnopqrstuvwxyz',
               rpad('a',36,'a'))

```

将所有的数字和字母转换为一个特定的字母（在这里选择字母“a”）。当数据转换完成后，含有真正的字母与数字的所有字符串都变成由同一个字母组成的字符串（在本例中，全是由字母“a”组成）。单独执行一下 TRANSLATE 函数就可以看出这一点：

```

select data, translate(lower(data),
                       '0123456789abcdefghijklmnopqrstuvwxyz',
                       rpad('a',36,'a'))
   from V

```

DATA	TRANSLATE (LOWER (DATA))
CLARK	aaaaa
...	
SMITH, \$800.00	aaaaa, \$aaa.aa
...	
ALLEN30	aaaaaaa
...	

所有字母或数字都被转换掉了，但是字符串的长度并没有改变。因为长度是相同的，要保留的就是那些调用 TRANSLATE 返回结果全为 a 的行。

```
select data, translate(lower(data),
                        '0123456789abcdefghijklmnopqrstuvwxyz',
                        rpad('a',36,'a')) translated,
       rpad('a',length(data),'a') fixed
from v
```

DATA	TRANSLATED	FIXED
CLARK	aaaaa	aaaaa
...		
SMITH, \$800.00	aaaaa, \$aaa.aa	aaaaaaaaaaaaaaa
...		
ALLEN30	aaaaaaa	aaaaaaa
...		

最后一步操作就是保留 TRANSLATED 与 FIXED 列中值相等的行。

## MySQL

在 WHERE 子句中的表达式：

```
where data regexp '^[^0-9a-zA-Z]' = 0
```

使得只包含数字或字符的行返回。在括号中的范围“0-9a-zA-Z”，表示所有数字和字母，字符“^”表示否定的意思，所以这个表达式的意思就是“非数字或字母”。返回值为1表示真，而0表示假，所以，整个表达式的意思就是“返回除数字和字母外还包含其他数据这一条件为假的行”。

## SQL Server

第一步就是遍历由视图 V 返回的每一行，由 DATA 中每个字符都作为一行返回，由 C 的值代表由 DATA 中每个单独的字符。

data	pos	c	val
ADAMS, \$1100.00	1	A	65
ADAMS, \$1100.00	2	D	68
ADAMS, \$1100.00	3	A	65
ADAMS, \$1100.00	4	M	77
ADAMS, \$1100.00	5	S	83
ADAMS, \$1100.00	6	,	44
ADAMS, \$1100.00	7		32
ADAMS, \$1100.00	8	\$	36
ADAMS, \$1100.00	9	1	49
ADAMS, \$1100.00	10	1	49
ADAMS, \$1100.00	11	0	48
ADAMS, \$1100.00	12	0	48
ADAMS, \$1100.00	13	.	46
ADAMS, \$1100.00	14	0	48
ADAMS, \$1100.00	15	0	48

内联视图 Z 并不只是一行一行地返回在 DATA 列中的每个字符，也给出各字符的 ASCII 值。在 SQL Server 的这个特定版本中，范围在 48~122 之间的值表示数字或字母字符。根据这些知识，可以对 DATA 中的每行进行分组，并且筛选掉那些 ASCII 最小值不在 48~122 范围之内的行。

## 6.7 提取姓名的大写首字母缩写问题

将全名转换为大写首字母缩写。考虑下面的名字：

Stewie Griffin

要返回如下结果：

S.G.

### 解决方案

需要注意的就是SQL并不像C或Python语言那样灵活，所以，创建一个通用的解决方案来处理所有格式的姓名对于SQL来说不是一件容易的事情。在这里所介绍的解决方案假定名字都是由名和姓，或是由名、中间名（中间名缩写）及姓组成。

#### DB2

使用内置函数 REPLACE、TRANSLATE 和 REPEAT 来提取出大写首字母缩写：

```

1 select replace(
2     replace(
3         translate(replace('Stewie Griffin', '.', ''),
4             repeat('#',26),
5             'abcdefghijklmnopqrstuvwxyz'),
6         '#',''), ' ','.')
7     || '.'
8 from t1

```

#### MySQL

使用内置函数 CONCAT、CONCAT\_WS、SUBSTRING 和 SUBSTRING\_INDEX 来提取出大写首字母缩写：

```

1 select case
2     when cnt = 2 then
3         trim(trailing '.' from
4             concat_ws('.',
5                 substr(substring_index(name, ' ',1),1,1),
6                 substr(name,
7                     length(substring_index(name, ' ',1))+2,1),
8                 substr(substring_index(name, ' ',-1),1,1),
9                 '.'))
10    else
11        trim(trailing '.' from
12            concat_ws('.',
13                substr(substring_index(name, ' ',1),1,1),
14                substr(substring_index(name, ' ',-1),1,1)
15            ))
16    end as initials
17 from (
18 select name,length(name)-length(replace(name, ' ','')) as cnt
19 from (
20 select replace('Stewie Griffin',' ','') as name from t1
21 )y
22 )x

```

## Oracle 和 PostgreSQL

使用内置函数 REPLACE、TRANSLATE 和 RPAD 来提取出大写首字母缩写：

```

1 select replace(
2     replace(
3         translate(replace('Stewie Griffin', '.', ''),
4                     'abcdefghijklmnopqrstuvwxyz',
5                     rpad('#',26,'#') ), '#', '') , ' ', '.' ) || '.'
6   from t1

```

## SQL Server

在本书写作的时候，SQL Server 不支持 TRANSLATE 及 CONCAT\_WS 函数。

## 讨论

通过分离出大写字母就可以提取姓名中的大写首字母缩写。下面详细介绍每个 DBMS 的解决方案。

## DB2

REPLACE 函数删除姓名中的所有点（用来处理中间名的缩写），然后，TRANSLATE 函数将所有的小写字母转换为“#”。

```

select translate(replace('Stewie Griffin', '.', ''),
                repeat('#',26),
                'abcdefghijklmnopqrstuvwxyz')
  from t1

TRANSLATE('STE
-----
S##### G#####

```

在此时，除了大写首字母之外，所有的字符全部为#，使用 REPLACE 函数来删除所有的# 字符。

```

select replace(
    translate(replace('Stewie Griffin', '.', ''),
              repeat('#',26),
              'abcdefghijklmnopqrstuvwxyz'), '#', '')
  from t1

REP
---
S G

```

下一步是再次使用 REPLACE 函数将所有空格替换为句点：

```

select replace(
    replace(
        translate(replace('Stewie Griffin', '.', ''),
                  repeat('#',26),
                  'abcdefghijklmnopqrstuvwxyz'), '#', ''), ' ', '.') || '.'
  from t1

REPLA
-----
S.G

```

最后一步就是在大写首字母缩写结果后面附加一个点。

## Oracle 和 PostgreSQL

REPLACE函数将删除在姓名中所有的点(用来处理中间名的缩写),然后,TRANSLATE函数将所有的小写字母转换为“#”。

```
select translate(replace('Stewie Griffin','.', ''),
                'abcdefghijklmnopqrstuvwxyz',
                rpad('#',26,'#'))
from t1
TRANSLATE('STE
-----
S##### G#####
```

此时,除了字首大写字母之外,所有的字符全部为#,使用REPLACE函数来删除所有的#字符。

```
select replace(
    translate(replace('Stewie Griffin','.', ''),
              'abcdefghijklmnopqrstuvwxyz',
              rpad('#',26,'#')), '#', '')
from t1
REP
---
S G
```

下一步是再次使用REPLACE函数将所有的空格替换为句点:

```
select replace(
    replace(
        translate(replace('Stewie Griffin','.', ''),
                  'abcdefghijklmnopqrstuvwxyz',
                  rpad('#',26,'#')), '#', ''), ' ', '.') || '.'
from t1
REPLA
-----
S.G
```

最后一步就是在大写首字母缩写结果后面附加一个点。

## MySQL

内联视图Y用来删除姓名中所有的句点,内联视图X查找姓名中空格的总数,这样就可以按照这个总数来确定调用SUBSTR函数的次数,提取出字首大写字母。3次调用SUBSTRING\_INDEX函数,分别根据空格的位置将字符串解析为姓名的各部分;因为在这个示例中只有名和姓,所以会执行case语句的ELSE部分:

```
select substr(substring_index(name, ' ',1),1,1) as a,
       substr(substring_index(name, ' ',-1),1,1) as b
from (select 'Stewie Griffin' as name from t1) x
A B
--
S G
```

如果此姓名包含有中间名或是中间名缩写,执行下面的语句就可以返回其字首大写字母

```
substr(name,length(substring_index(name, ' ',1))+2,1)
```

该语句查找到名的结束位置,然后向后移动两格到中间名或中间名缩写的开始位置,这也是SUBSTR函数的开始位置。因为只要保留一个字符,中间名或中间大写字母就会被成功地返回。字首大写字母传送到CONCAT\_WS函数,该函数用句点将字首大写字母隔开。

```
select concat_ws('.',
                 substr(substring_index(name, ' ',1),1,1),
                 substr(substring_index(name, ' ',-1),1,1),
                 '.' ) a
  from (select 'Stewie Griffin' as name from t1) x

A
-----
S.G..
```

最后一步就是清理掉多余的句点。

## 6.8 按字符串中的部分内容排序

### 问题

按照字符串的子串来对结果集排序。考虑下面列出的记录:

```
ENAME
-----
SMITH
ALLEN
WARD
JONES
MARTIN
BLAKE
CLARK
SCOTT
KING
TURNER
ADAMS
JAMES
FORD
MILLER
```

要对这些记录按照每个姓名的最后两个字符排序:

```
ENAME
-----
ALLEN
TURNER
MILLER
JONES
JAMES
MARTIN
BLAKE
ADAMS
KING
WARD
FORD
CLARK
SMITH
```

SCOTT

## 解决方案

这种解决方案的关键就是找到并使用相应DBMS的内置函数来提取出要据以进行排序的子字符串，代表性的函数就是 SUBSTR 函数。

## DB2, Oracle、MySQL 和 PostgreSQL

联合使用内置函数 LENGTH 和 SUBSTR 来按照字符串的指定部分进行排序：

```
1 select ename
2   from emp
3  order by substr(ename,length(ename)-1,2)
```

## SQL Server

使用 SUBSTRING 和 LEN 来按照字符串的指定部分进行排序：

```
1 select ename
2   from emp
3  order by substring(ename,len(ename)-1,2)
```

## 讨论

通过在 ORDER BY 子句中使用 SUBSTR 表达式，可以使用字符串中的任何部分来对结果集进行排序。不仅限于 SUBSTR 函数，几乎可以按任何表达式的结果来排序。

## 6.9 按字符串中的数值排序

### 问题

基于字符串中的数值来排序结果集。考虑下面的视图：

```
create view V as
select e.ename ||' '||
       cast(e.empno as char(4))||' '||
       d.dname as data
  from emp e, dept d
 where e.deptno=d.deptno
```

此视图返回下列数据：

```
DATA
-----
CLARK  7782 ACCOUNTING
KING   7839 ACCOUNTING
MILLER 7934 ACCOUNTING
SMITH  7369 RESEARCH
JONES  7566 RESEARCH
SCOTT  7788 RESEARCH
ADAMS  7876 RESEARCH
FORD   7902 RESEARCH
ALLEN  7499 SALES
WARD   7521 SALES
MARTIN 7654 SALES
BLAKE  7698 SALES
```



```
TURNER 7844 SALES
JAMES 7900 SALES
```

要根据员工编号给出结果排序，该编号位于员工姓名和其部门之间。

```
DATA
-----
SMITH 7369 RESEARCH
ALLEN 7499 SALES
WARD 7521 SALES
JONES 7566 RESEARCH
MARTIN 7654 SALES
BLAKE 7698 SALES
CLARK 7782 ACCOUNTING
SCOTT 7788 RESEARCH
KING 7839 ACCOUNTING
TURNER 7844 SALES
ADAMS 7876 RESEARCH
JAMES 7900 SALES
FORD 7902 RESEARCH
MILLER 7934 ACCOUNTING
```

## 解决方案

不同的DBMS中的解决方案使用的函数与语法都不尽相同，但是方法（使用内置的函数REPLACE和TRANSLATE）是相同的。具体就是使用REPLACE和TRANSLATE来从字符串中删除非数字字符，只留下用于排序的数字值。

### DB2

使用内置函数REPLACE和TRANSLATE来根据字符串中的数字字符排序：

```
1 select data
2   from V
3   order by
4     cast(
5       replace(
6         translate(data,repeat('#',length(data)),
7         replace(
8           translate(data,'#####','0123456789'),
9           '#','')), '#','') as integer)
```

### Oracle

使用内置函数REPLACE和TRANSLATE来根据字符串中的数字字符排序：

```
1 select data
2   from V
3   order by
4     to_number(
5       replace(
6         translate(data,
7         replace(
8           translate(data,'0123456789','#####'),
9           '#'),rpad('#',20,'#')),'#'))
```

### PostgreSQL

使用内置函数REPLACE和TRANSLATE来根据字符串中的数字字符排序：

```

1 select data
2   from V
3  order by
4     cast(
5       replace(
6         translate(data,
7           replace(
8             translate(data, '0123456789', '#####'),
9             '#', ''), rpad('#', 20, '#')), '#', '') as integer)

```

## MySQL 和 SQL Server

在本书写作的时候，两者都不支持 TRANSLATE 函数。

## 讨论

视图 V 的目的就是为演示本节的解决方案提供数据。该视图只包含 TMP 表中的几列。本解决方案显示了如何使用这样的连接在一起的文本作为输入数据，并根据嵌入在文本中的员工号码来进行排序。

各解决方案中的 ORDER BY 子句看起来有些复杂，但是执行起来的效果很好，并且它每一部分所执行的操作都很直接。要按照字符串中的数字排序，首先要删除所有的非数字字符，当它们全部被删除之后，就只剩下数字字符，最后就可以对它们进行排序。在检查每个函数调用之前，重要的是理解每个函数调用的顺序。现在从最里面的函数调用开始，TRANSLATE（在每个原始解决方案的第 8 行），可以看到：

1. 调用 TRANSLATE（第 8 行），并且其结果返回到
2. 调用 REPLACE（第 7 行），并且这些结果返回到
3. 调用 TRANSLATE（第 6 行），并且这些结果返回到
4. 调用 REPLACE（第 5 行），并且这些结果被最终返回
5. 转换为数值

第一步是将数字转换为字符，所要被转换成的字符必须是在这些字符串中没有的字符。在此示例中选择“#”，并且使用 TRANSLATE 来将所有的（这里原文有误，译者注）数字字符转换为“#”字符。例如，下列查询的左侧显示了原始数据，而右侧显示了第一次转换后的结果：

```

select data,
       translate(data, '0123456789', '#####') as tmp
  from V

```

DATA			TMP		
CLARK	7782	ACCOUNTING	CLARK	####	ACCOUNTING
KING	7839	ACCOUNTING	KING	####	ACCOUNTING
MILLER	7934	ACCOUNTING	MILLER	####	ACCOUNTING
SMITH	7369	RESEARCH	SMITH	####	RESEARCH
JONES	7566	RESEARCH	JONES	####	RESEARCH

SCOTT	7788	RESEARCH	SCOTT	####	RESEARCH
ADAMS	7876	RESEARCH	ADAMS	####	RESEARCH
FORD	7902	RESEARCH	FORD	####	RESEARCH
ALLEN	7499	SALES	ALLEN	####	SALES
WARD	7521	SALES	WARD	####	SALES
MARTIN	7654	SALES	MARTIN	####	SALES
BLAKE	7698	SALES	BLAKE	####	SALES
TURNER	7844	SALES	TURNER	####	SALES
JAMES	7900	SALES	JAMES	####	SALES

TRANSLATE 函数在每个字符串中查找数字，并且将每个数字转换为“#”字符。编辑完成后的字符串返回到 REPLACE 函数（第 11 行），此函数将删除所有的“#”字符。

```
select data,
replace(
translate(data,'0123456789','#####'),'#') as tmp
from v
```

DATA			TMP		
CLARK	7782	ACCOUNTING	CLARK		ACCOUNTING
KING	7839	ACCOUNTING	KING		ACCOUNTING
MILLER	7934	ACCOUNTING	MILLER		ACCOUNTING
SMITH	7369	RESEARCH	SMITH		RESEARCH
JONES	7566	RESEARCH	JONES		RESEARCH
SCOTT	7788	RESEARCH	SCOTT		RESEARCH
ADAMS	7876	RESEARCH	ADAMS		RESEARCH
FORD	7902	RESEARCH	FORD		RESEARCH
ALLEN	7499	SALES	ALLEN		SALES
WARD	7521	SALES	WARD		SALES
MARTIN	7654	SALES	MARTIN		SALES
BLAKE	7698	SALES	BLAKE		SALES
TURNER	7844	SALES	TURNER		SALES
JAMES	7900	SALES	JAMES		SALES

然后，这些字符串再次返回给 TRANSLATE 函数，这是在这个解决方案中第二次调用（最外面的）TRANSLATE 函数。TRANSLATE 函数在表 TMP 中查找与剩余字符相匹配的字符，如果找到，它们也将转换为“#”字符。这次转换就是将所有的非数字字符作为同一个字符对待（因为这些都转换为同一个字符）。

```
select data, translate(data,
replace(
translate(data,'0123456789','#####'),
'#'),
rpad('#',length(data),'#')) as tmp
from v
```

DATA			TMP
CLARK	7782	ACCOUNTING	#####7782#####
KING	7839	ACCOUNTING	#####7839#####
MILLER	7934	ACCOUNTING	#####7934#####
SMITH	7369	RESEARCH	#####7369#####
JONES	7566	RESEARCH	#####7566#####
SCOTT	7788	RESEARCH	#####7788#####
ADAMS	7876	RESEARCH	#####7876#####
FORD	7902	RESEARCH	#####7902#####
ALLEN	7499	SALES	#####7499#####
WARD	7521	SALES	#####7521#####
MARTIN	7654	SALES	#####7654#####
BLAKE	7698	SALES	#####7698#####
TURNER	7844	SALES	#####7844#####
JAMES	7900	SALES	#####7900#####

下一步通过调用 REPLACE 函数（第 8 行）来删除所有的“#”字符，只剩下数字：

```
select data, replace(
    translate(data,
        replace(
            translate(data, '0123456789', '#####'),
            '#'),
        rpad('#', length(data), '#')), '#') as tmp
from v
```

DATA			TMP
CLARK	7782	ACCOUNTING	7782
KING	7839	ACCOUNTING	7839
MILLER	7934	ACCOUNTING	7934
SMITH	7369	RESEARCH	7369
JONES	7566	RESEARCH	7566
SCOTT	7788	RESEARCH	7788
ADAMS	7876	RESEARCH	7876
FORD	7902	RESEARCH	7902
ALLEN	7499	SALES	7499
WARD	7521	SALES	7521
MARTIN	7654	SALES	7654
BLAKE	7698	SALES	7698
TURNER	7844	SALES	7844
JAMES	7900	SALES	7900

最后，使用 DBMS 的适当函数（常常是 CAST）将 TMP 转换为数值（第 4 行），并结束此操作：

```
select data, to_number(
    replace(
        translate(data,
            replace(
                translate(data, '0123456789', '#####'),
                '#'),
            rpad('#', length(data), '#')), '#')) as tmp
from v
```

DATA			TMP
CLARK	7782	ACCOUNTING	7782
KING	7839	ACCOUNTING	7839
MILLER	7934	ACCOUNTING	7934
SMITH	7369	RESEARCH	7369
JONES	7566	RESEARCH	7566
SCOTT	7788	RESEARCH	7788
ADAMS	7876	RESEARCH	7876
FORD	7902	RESEARCH	7902
ALLEN	7499	SALES	7499
WARD	7521	SALES	7521
MARTIN	7654	SALES	7654
BLAKE	7698	SALES	7698
TURNER	7844	SALES	7844
JAMES	7900	SALES	7900

在开发这样的查询时，将这些表达式先放到 SELECT 列表中会很有帮助。通过这种方法，可以随时看到通往最终解决方案的过程中的中间结果。然而，因为本章的目的是对结果集排序，所以必须将所有这些函数调用放到 ORDER BY 子句中。

```
select data
from v
order by
    to_number(
```

```

      replace(
translate( data,
      replace(
translate( data,'0123456789','#####'),
      '#'),rpad('#',length(data),'#'),'#'))

```

DATA

```

-----
SMITH  7369 RESEARCH
ALLEN   7499 SALES
WARD    7521 SALES
JONES   7566 RESEARCH
MARTIN  7654 SALES
BLAKE   7698 SALES
CLARK   7782 ACCOUNTING
SCOTT   7788 RESEARCH
KING    7839 ACCOUNTING
TURNER  7844 SALES
ADAMS   7876 RESEARCH
JAMES   7900 SALES
FORD    7902 RESEARCH
MILLER  7934 ACCOUNTING

```

最后需要注意的就是，在组成该视图数据的3个字段中，只有一个字段是数字。如果该视图数据包含多个数字字段，则在排序之前，所有的数字字段都连接到一起，变成一个数值。

## 6.10 根据表中的行创建一个分隔列表

### 问题

要求将表中的行作为在一个分隔列表中的值，分界符可能是逗号而不是通常使用的竖线。例如，要将如下的数据集：

```

DEPTNO EMPS
-----
10 CLARK
10 KING
10 MILLER
20 SMITH
20 ADAMS
20 FORD
20 SCOTT
20 JONES
30 ALLEN
30 BLAKE
30 MARTIN
30 JAMES
30 TURNER
30 WARD

```

转换为下列内容：

```

DEPTNO EMPS
-----
10 CLARK,KING,MILLER
20 SMITH,JONES,SCOTT,ADAMS,FORD
30 ALLEN,WARD,MARTIN,BLAKE,TURNER,JAMES

```

## 解决方案

对于这个问题,每个DBMS都有其不同的解决方法。关键就在于灵活运用不同的DBMS所提供的内置函数。在了解DBMS能够提供什么后,就可以充分挖掘DBMS的功能,进而找到创造性的解决方案来解决SQL中一般难以解决的问题。

### DB2

使用递归的 WITH 函数来构建此分界列表:

```

1  with x (deptno, cnt, list, empno, len)
2  as (
3  select deptno, count(*) over (partition by deptno),
4         cast(ename as varchar(100)), empno, 1
5  from emp
6  union all
7  select x.deptno, x.cnt, x.list || ',' || e.ename, e.empno, x.len+1
8  from emp e, x
9  where e.deptno = x.deptno
10     and e.empno > x.empno
11     )
12 select deptno, list
13 from x
14 where len = cnt

```

### MySQL

使用内置函数 GROUP\_CONCAT 来构建分隔列表:

```

1 select deptno,
2        group_concat(ename order by empno separator, ',') as emps
3 from emp
4 group by deptno

```

### Oracle

使用内置函数 SYS\_CONNECT\_BY\_PATH 来构建分隔列表:

```

1 select deptno,
2        ltrim(sys_connect_by_path(ename, ','), ',') emps
3 from (
4 select deptno,
5        ename,
6        row_number( ) over
7              (partition by deptno order by empno) rn,
8        count(*) over
9              (partition by deptno) cnt
10 from emp
11 )
12 where level = cnt
13 start with rn = 1
14 connect by prior deptno = deptno and prior rn = rn-1

```

### PostgreSQL

PostgreSQL不提供用来创建分隔列表的标准内置函数,所以它必须预先知道在这个列表中将有多少值。一旦知道最大列表的大小,就可以判断需要添加的项数,从而用标准的变换和连接操作来创建分隔列表。

```

1 select deptno,
2     rtrim(
3         max(case when pos=1 then emps else '' end)||
4         max(case when pos=2 then emps else '' end)||
5         max(case when pos=3 then emps else '' end)||
6         max(case when pos=4 then emps else '' end)||
7         max(case when pos=5 then emps else '' end)||
8         max(case when pos=6 then emps else '' end),'',
9     ) as emps
10 from (
11 select a.deptno,
12        a.ename||',' as emps,
13        d.cnt,
14        (select count(*) from emp b
15         where a.deptno=b.deptno and b.empno <= a.empno) as pos
16 from emp a,
17      (select deptno, count(ename) as cnt
18       from emp
19       group by deptno) d
20 where d.deptno=a.deptno
21      ) x
22 group by deptno
23 order by 1

```

## SQL Server

使用递归的 WITH 函数来构建分界列表：

```

1  with x (deptno, cnt, list, empno, len)
2  as (
3  select deptno, count(*) over (partition by deptno),
4         cast(ename as varchar(100)),
5         empno,
6         1
7  from emp
9  union all
9  select x.deptno, x.cnt,
10        cast(x.list + ',' + e.ename as varchar(100)),
11        e.empno, x.len+1
12  from emp e, x
13  where e.deptno = x.deptno
14        and e.empno > x. empno
15        )
16 select deptno,list
17 from x
18 where len = cnt
19 order by 1

```

## 讨论

在 SQL 中创建分隔列表是非常有用的，因为这是一个有共性的需求。但是每个 DBMS 都有其独特的方法在 SQL 中构建这样的列表，这些特别的解决方案之间共性很少，所使用的技术有递归、层次函数、标准转换操作和聚集等。

## DB2 和 SQL Server

对于这两个数据库来说，它们相应的解决方案只是在语法上有些不同（DB2 的串连接操作符为“||”，而 SQL Server 的是“+”），但其所用的方法是相同的。WITH 子句中的第一个查询（在 UNION ALL 子句上面的部分）返回每个员工的下列信息：部门、部分中的员工数、姓名、ID 号和常量 1（在这里没有任何作用），在第二个查询中（在 UNION

ALL子句下面的部分)用递归构建此列表。要理解此列表是如何构建的,看看以下从解决方案中节选的部分语句:首先,第二个查询的SELECT列表中的第三项:

```
x.list || ',' || e.ename
```

然后看看同一查询的WHERE子句:

```
where e.deptno = x.deptno
and e.empno > x.empno
```

首先,此解决方案确保员工都在相同的部门;然后,对UNION ALL上半部分返回的每个员工,将EMPNO大于自身的所有员工姓名添到自己后面,这样,可以确保所有员工都不会将自己的姓名加到自己后面。每处理一个员工后表达式

```
x.len+1
```

将LEN(初始值为1)值增加1。当增加后的值等于所在部门的员工人数时,

```
where len = cnt
```

就表示已经处理过所有的员工,此列表已经构建完成。这对此查询非常重要,因为它不仅是列表构建完成的信号,而且及时停止递归以避免不必要的运行。

## MySQL

函数GROUP\_CONCAT完成所有的操作,它将传给它的列(本例中为ENAME)中的所有值连接起来。这是一个聚集函数,因此查询中需要有GROUP BY子句。

## Oracle

要理解Oracle查询,首先要对它进行拆分。单独运行内联视图(4~10行),就可以得到每个员工的如下信息:所在部门、姓名、按EMPNO升序在部门中的序号以及部门员工数。例如:

```
select deptno,
       ename,
       row_number( ) over
         (partition by deptno order by empno) rn,
       count(*) over (partition by deptno) cnt
from emp
```

DEPTNO	ENAME	RN	CNT
10	CLARK	1	3
10	KING	2	3
10	MILLER	3	3
20	SMITH	1	5
20	JONES	2	5
20	SCOTT	3	5
20	ADAMS	4	5
20	FORD	5	5
30	ALLEN	1	6
30	WARD	2	6
30	MARTIN	3	6
30	BLAKE	4	6



```

30 TURNER      5    6
30 JAMES       6    6

```

序号（在查询中别名为RN）的目的是可以用来作为遍历树。因为函数ROW\_NUMBER可以从1开始产生没有重复或间隔的顺序号，只要减一（从当前值）就可以引用前一（或父）行。例如，3前面的数字就是3减1等于2，这里，2就是3的父行，这从第12行就可以看出来。此外，以下两行

```

start with rn = 1
connect by prior deptno = deptno

```

根据RN等于1来确定每个DEPTNO的根记录，并为每个新遇到的部门（只要RN被发现等于1时）创建一个新的列表。

此时必须停一下，看看ROW\_NUMBER函数的ORDER BY部分。注意，姓名是按照EMPNO排序号的，并且创建列表时也是按照这个顺序。每个部门中员工的人数（别名为CNT）也被计算出来，用来确认查询只返回包含该部门全部员工姓名的列表。这样做是因为SYS\_CONNECT\_BY\_PATH用迭代方式构建此列表，而最终结果也不能是不完整的列表。对于层次式的查询，伪列LEVEL从1开始（对于没有使用CONNECT BY的查询，LEVEL是0；而对10g或以上的产品，只有使用CONNECT BY时才能用LEVEL），并且每处理部门中的一个员工后该值增加1（代表层次中的深度）。因此，一旦LEVEL值等于CNT时，就表示已经遇到最后一个EMPNO，该列表已经完成。

---

**注意：** SYS\_CONNECT\_BY\_PATH函数会在列表最前面增加一个所选定的分隔符（这里是逗号）。用户可能需要也可能不需要这样的结果。在本章的解决方案中，调用LTRIM函数来删除列表最前面的逗号。

---

## PostgreSQL

PostgreSQL的解决方案需要预先知道在每个部门中员工的人数。单独运行内联视图（第11~18行）会生成一个结果集（对每个员工），其中包括该员工的部门、附加了逗号的姓名、在该员工部门中的员工人数及EMPNO小于该员工编号的人数：

deptno	emps	cnt	pos
20	SMITH,	5	1
30	ALLEN,	6	1
30	WARD,	6	2
20	JONES,	5	2
30	MARTIN,	6	3
30	BLAKE,	6	4
10	CLARK,	3	1
20	SCOTT,	5	3
10	KING,	3	2
30	TURNER,	6	5
20	ADAMS,	5	4
30	JAMES,	6	6
20	FORD,	5	5
10	MILLER,	3	3

标量子查询POS（第14~15行）用来根据EMPNO给每个员工排序号，例如，下面这行

```
max(case when pos = 1 then ename else '' end)||
```

计算POS的值是否等于1，如果POS等于1，则CASE表达式返回员工姓名，否则，返回NULL值。

首先要从表中查找出列表的最大可能项数。根据EMP表的数据，所有部门中员工人数最多有6人，所以在列表中项数最多为6。

下一步是开始创建列表，这通过对内联视图中返回的每一行执行若干条件逻辑（用CASE表达式）来实现最多有多少项连接到一起，就要写多少个CASE表达式。

如果POS等于1，就把当前姓名加入到列表中；第二个CASE表达式判断POS是否等于2，如果等于，那么就将第二个姓名加入到第一个姓名之后，如果没有第二个姓名，那么会向第一个姓名之后添加一个逗号（此过程对POS的每个值都要重复一遍，直到最后一个值）。

MAX函数必须使用，因为对每个部门来说只构建一个列表（也可以使用MIN函数，在此情况下作用是一样的，因为每次计算case表达式，POS只返回一个值）。只要使用聚集函数，SELECT列表中没有被聚集函数作用的所有项，都必须加进GROUP BY子句中，这样就可以保证SELECT列表中没有被聚集函数作用的项，每项只有一行。

注意，还需要RTRIM函数来删除最后的逗号，逗号的数目总应该等于列表的最大项数（本例中为6）。

## 6.11 将分隔数据转换为多值IN列表

### 问题

已经有了分隔数据，想要将其转换为WHERE子句IN列表中的项目。考虑下面的字符串：

```
7654,7698,7782,7788
```

要将该字符串用在WHERE子句中，但是下面的SQL语句是错误的，因为EMPNO是一个数值列：

```
select ename,sal,deptno
from emp
where empno in ( '7654,7698,7782,7788' )
```

因为EMPNO是一个数值列，而此IN列表是一个字符串值，所以此SQL语句会失败。现要将此字符串转换为用逗号分解的数值列表。

### 解决方案

表面上看SQL应该将分隔字符串作为一个分隔值列表对待，但是实际情况不是这样。当

SQL 遇到括在引号中的逗号时，并不知道此符号表示多值列表，SQL 必须将括在引号中的内容作为一个整体对待，也就是一个字符串值。因此必须将字符串分解为各个单独的 EMPNO。这种解决方案的关键就是需要遍历字符串，但并不是一个字符一个字符地遍历，而是要将这个字符串转换为有效的 EMPNO 值。

## DB2

通过遍历传递给 IN 列表的字符串，可以轻松地将其转换为若干行。在这里函数 ROW\_NUMBER、LOCATE 和 SUBSTR 尤其有用：

```
1 select empno,ename,sal,deptno
2   from emp
3   where empno in (
4     select cast(substr(c,2,locate(',',c,2)-2) as integer) empno
5     from (
6       select substr(csv.emps,cast(iter.pos as integer)) as c
7         from (select '||'7654,7698,7782,7788'||',' emp
8              from t1) csv,
9              (select id as pos
10               from t100 ) iter
11      where iter.pos <= length(csv.emps)
12            ) x
13      where length(c) > 1
14            and substr(c,1,1) = ','
15            ) y
```

## MySQL

通过遍历传递给 IN 列表的字符串，可以轻松地将其转换为若干行：

```
1 select empno, ename, sal, deptno
2   from emp
3   where empno in
4     (
5     select substring_index(
6       substring_index(list.vals,',',iter.pos),',',-1) empno
6     from (select id pos from t10) as iter,
7          (select '7654,7698,7782,7788' as vals
8           from t1) list
9     where iter.pos <=
10           (length(list.vals)-length(replace(list.vals,',','')))+1
11           ) x
```

## Oracle

通过遍历传递给 IN 列表的字符串，可以轻松地将其转换为若干行。这里函数 ROWNUM，SUBSTR 和 INSTR 尤其有用：

```
1 select empno,ename,sal,deptno
2   from emp
3   where empno in (
4     select to_number(
5       rtrim(
6         substr(emps,
7           instr(emps,',',1,iter.pos)+1,
8           instr(emps,',',1,iter.pos+1) -
9           instr(emps,',',1,iter.pos)),',') emps
10      from (select '||'7654,7698,7782,7788'||',' emps from t1) csv,
11            (select rownum pos from emp) iter
```

```

12         where iter.pos <= ((length(csv.emps)-
13                             length(replace(csv.emps,', ')))/length(',')-1
14     )

```

## Postgres

通过遍历传递给 IN 列别的字符串，可以很轻松地将其转换为若干行。使用函数 SPLIT\_PART 可以简化将字符串解析为单独的数值列表的工作：

```

1 select ename,sal,deptno
2   from emp
3  where empno in (
4 select cast(empno as integer) as empno
5   from (
6 select split_part(list.vals,',',iter.pos) as empno
7   from (select id as pos from t10) iter,
8        (select ','||'7654,7698,7782,7788'||',' as vals
9         from t1) list
10  where iter.pos <=
11        length(list.vals)-length(replace(list.vals,', '))
12        ) z
13  where length(empno) > 0
14        ) x

```

## SQL Server

通过遍历传递给 IN 列别的字符串，可以很轻松地将其转换为若干行。这里函数 ROW\_NUMBER、CHARINDEX 和 SUBSTRING 尤其有用：

```

1 select empno,ename,sal,deptno
2   from emp
3  where empno in (select substring(c,2,charindex(',' ,c,2)-2) as empno
4   from (
5 select substring(csv.emps,iter.pos,len(csv.emps)) as c
6   from (select ','+'7654,7698,7782,7788'+',' as emps
7         from t1) csv,
8        (select id as pos
9         from t100) iter
10  where iter.pos <= len(csv.emps)
11        ) x
12  where len(c) > 1
13    and substring(c,1,1) = ','
14        ) y

```

## 讨论

这种解决方案中第一步，也是最重要的一步就是遍历字符串。一旦完成了这步操作，剩下的操作就是使用 DBMS 提供的函数来将字符串解析为单独的数值。

## DB2 和 SQL Server

内联视图 X (第 6~11 行) 遍历字符串，这里用的是“穿越”字符串的思想，所以其每一行都比其上一行少一个字符：

```

,7654,7698,7782,7788,
7654,7698,7782,7788,
654,7698,7782,7788,
54,7698,7782,7788,
4,7698,7782,7788,

```

```
,7698,7782,7788,
7698,7782,7788,
698,7782,7788,
98,7782,7788,
8,7782,7788,
,7782,7788,
7782,7788,
782,7788,
82,7788,
2,7788,
,7788,
7788,
788,
88,
8,
,
```

注意，因为整个字符串是由逗号（分界符）括起来的，所以不需要特殊的检查来确定字符串从哪里开始及到哪里结束：

下一步是只保留要用于内部列表中的值。除了最后只有单独逗号的一行之外，保留以逗号开头的行。使用 SUBSTR 或 SUBSTRING 函数来识别哪些行以逗号开头，并保留在此行中到下一个逗号之间的所有字符。此操作完成后，将字符串转换为数值，这样就可以用来正确地跟数值列 EMPNO 运算了（行 4~14）：

```
EMPNO
-----
 7654
 7698
 7782
 7788
```

最后，使用在子查询中的结果来返回想要得到的行。

## MySQL

内联视图（行 5~9）遍历字符串。第 10 行的表达式通过查找逗号（分隔符）的数量来确定在字符串中有多少个值并加 1。函数 SUBSTRING\_INDEX（第 6 行）返回在字符串中第 n 个逗号（分隔符）之前（到它左边）的所有字符：

```
+-----+
| empno |
+-----+
| 7654   |
| 7654,7698 |
| 7654,7698,7782 |
| 7654,7698,7782,7788 |
+-----+
```

这些行随后传递给 SUBSTRING\_INDEX（第 5 行）函数的另一次调用，此时第 n 个分隔符参数为 -1，这表示在第 n 个分隔符右边所有的值需要保留。

```
+-----+
| empno |
+-----+
| 7654   |
| 7698   |
+-----+
```

```
| 7782 |
| 7788 |
+-----+
```

最后一步是将结果加入到一个子查询中。

## Oracle

第一步是遍历字符串：

```
select emps,pos
  from (select '||'7654,7698,7782,7788'||',' emps
        from t1) csv,
        (select rownum pos from emp) iter
 where iter.pos <=
        ((length(csv.emps)-length(replace(csv.emps,',')))/length(',')-1
```

EMPS	POS
,7654,7698,7782,7788,	1
,7654,7698,7782,7788,	2
,7654,7698,7782,7788,	3
,7654,7698,7782,7788,	4

有多少行返回，就表示在列表中有多少个值。POS 值对于查询是至关重要的，因为要用它来将字符串分解为单独的值。使用 SUBSTR 和 INSTR 函数来分解字符串，POS 用来确定在每个字符串中第 n 个分隔符的位置。由于字符串是用逗号括起来的，所以不需要特殊的检测手段来确定字符串的开始位置与结束位置。传递给 SUBSTR，INSTR（7~9 行）确定第 n 个和第 n+1 个分隔符的位置。用下一个逗号的位置值（字符串中下一个逗号的位置）减去当前逗号的位置值（字符串中当前逗号的位置），就可以从字符串中提取出每个值。

```
select substr(emps,
             instr(emps,',',1,iter.pos)+1,
             instr(emps,',',1,iter.pos+1) -
             instr(emps,',',1,iter.pos)) emps
  from (select '||'7654,7698,7782,7788'||',' emps
        from t1) csv,
        (select rownum pos from emp) iter
 where iter.pos <=
        ((length(csv.emps)-length(replace(csv.emps,',')))/length(',')-1
```

EMPS
7654,
7698,
7782,
7788,

最后一步就是删除每个值后面的逗号，将其转换为数值并且将其插入到子查询中。

## PostgreSQL

内联视图 Z（6~9 行）遍历字符串，返回行的数量也就是在字符串中有多少值。要找出在字符串中有多少值，用带有分隔符的字符串长度减去不带分隔符的字符串长度即可（第 9 行），函数 SPLIT\_PART 用来分解字符串，该函数查找第 n 个分隔符之前的值。

```

SELECT 列表.vals,
       split_part(list.vals,',',iter.pos) as empno,
       iter.pos
  from (select id as pos from t10) iter,
       (select '||'7654,7698,7782,7788'||',' as vals
        from t1) list
 where iter.pos <=
       length(list.vals)-length(replace(list.vals,',',''))

      vals      | empno | pos
-----+-----+---
,7654,7698,7782,7788, |      | 1
,7654,7698,7782,7788, | 7654  | 2
,7654,7698,7782,7788, | 7698  | 3
,7654,7698,7782,7788, | 7782  | 4
,7654,7698,7782,7788, | 7788  | 5

```

最后一步就是将返回值（EMPNO）转换为数值，并且插入到子查询中。

## 6.12 按字母顺序排列字符串

### 问题

对表中的字符串，按照字母顺序排列其中的各个字符。考虑下面列出的数据集：

```

ENAME
-----
ADAMS
ALLEN
BLAKE
CLARK
FORD
JAMES
JONES
KING
MARTIN
MILLER
SCOTT
SMITH
TURNER
WARD

```

需要结果如下所示：

OLD_NAME	NEW_NAME
ADAMS	AADMS
ALLEN	AELLN
BLAKE	ABEKL
CLARK	ACKLR
FORD	DFOR
JAMES	AEJMS
JONES	EJNOS
KING	GIKN
MARTIN	AIMNRT
MILLER	EILLMR
SCOTT	COSTT
SMITH	HIMST
TURNER	ENRRTU
WARD	ADRW

### 解决方案

该问题对解释为什么要理解所使用的DBMS及其提供的功能，是个很好的例子。如果所

使用的DBMS没有提供内置的函数来帮助实现这种解决方案,就需要使用创造性的方法,将MySQL的解决方案与其他的比较一下。

## DB2

要按照字母顺序排列各行字符串中的字符,需要遍历每个字符串,然后对其中的字符进行排序:

```

1  select ename,
2         max(case when pos=1 then c else '' end)||
3         max(case when pos=2 then c else '' end)||
4         max(case when pos=3 then c else '' end)||
5         max(case when pos=4 then c else '' end)||
6         max(case when pos=5 then c else '' end)||
7         max(case when pos=6 then c else '' end)
8  from (
9  select e.ename,
10         cast(substr(e.ename,iter.pos,1) as varchar(100)) c,
11         cast(row_number( )over(partition by e.ename
12                                order by substr(e.ename,iter.pos,1))
13            as integer) pos
14  from emp e,
15       (select cast(row_number( )over( ) as integer) pos
16        from emp) iter
17  where iter.pos <= length(e.ename)
18        ) x
19  group by ename

```

## MySQL

这里的关键就是GROUP\_CONCAT函数,该函数不仅将构成姓名的字符连接起来,还对它们进行排序:

```

1  select ename, group_concat(c order by c separator '')
2  from (
3  select ename, substr(a.ename,iter.pos,1) c
4  from emp a,
5       ( select id pos from t10 ) iter
6  where iter.pos <= length(a.ename)
7        ) x
8  group by ename

```

## Oracle

使用函数SYS\_CONNECT\_BY\_PATH允许以迭代方式构建一张列表:

```

1  select old_name, new_name
2  from (
3  select old_name, replace(sys_connect_by_path(c, ','), ',') new_name
4  from (
5  select e.ename old_name,
6         row_number( ) over(partition by e.ename
7                                order by substr(e.ename,iter.pos,1)) rn,
8         substr(e.ename,iter.pos,1) c
9  from emp e,
10       ( select rownum pos from emp ) iter
11  where iter.pos <= length(e.ename)
12  order by 1
13        ) x
14  start with rn = 1

```



```
15 connect by prior rn = rn-1 and prior old_name = old_name
16 )
17 where length(old_name) = length(new_name)
```

## PostgreSQL

PostgreSQL不支持任何内置的函数来对字符串中的字符进行排序,所以,必须在遍历每个字符串之外,还需要预先知道姓名的最大长度。在该种解决方案中,用视图V来增加它的可读性:

```
create or replace view V as
select x.*
  from (
select a.ename,
       substr(a.ename,iter.pos,1) as c
  from emp a,
       (select id as pos from t10) iter
 where iter.pos <= length(a.ename)
 order by 1,2
       ) x
```

下面列出的 select 语句利用了该视图:

```
1 select ename,
2       max(case when pos=1 then
3             case when cnt=1 then c
4                 else rpad(c,cast(cnt as integer),c)
5             end
6             else ''
7             end)||
8       max(case when pos=2 then
9             case when cnt=1 then c
10                else rpad(c,cast(cnt as integer),c)
11             end
12             else ''
13             end)||
14       max(case when pos=3 then
15             case when cnt=1 then c
16                 else rpad(c,cast(cnt as integer),c)
17             end
18             else ''
19             end)||
20       max(case when pos=4 then
21             case when cnt=1 then c
22                 else rpad(c,cast(cnt as integer),c)
23             end
24             else ''
25             end)||
26       max(case when pos=5 then
27             case when cnt=1 then c
28                 else rpad(c,cast(cnt as integer),c)
29             end
30             else ''
31             end)||
32       max(case when pos=6 then
33             case when cnt=1 then c
34                 else rpad(c,cast(cnt as integer),c)
35             end
36             else ''
37             end)
38  from (
39 select a.ename, a.c,
40       (select count(*)
41        from v b
```

```

42         where a.ename=b.ename and a.c=b.c ) as cnt,
43         (select count(*)+1
44         from v b
45         where a.ename=b.ename and b.c<a.c) as pos
46     from v a
47     ) x
48     group by ename

```

## SQL Server

要按照字母顺序排列各行字符串中的字符，需要遍历每个字符串，然后将这些字符排序：

```

1  select ename,
2         max(case when pos=1 then c else '' end)+
3         max(case when pos=2 then c else '' end)+
4         max(case when pos=3 then c else '' end)+
5         max(case when pos=4 then c else '' end)+
6         max(case when pos=5 then c else '' end)+
7         max(case when pos=6 then c else '' end)
8     from (
9     select e.ename,
10         substring(e.ename,iter.pos,1) as c,
11         row_number( ) over (
12             partition by e.ename
13             order by substring(e.ename,iter.pos,1)) as pos
14     from emp e,
15         (select row_number( )over(order by ename) as pos
16         from emp) iter
17     where iter.pos <= len(e.ename)
18     ) x
19     group by ename

```

## 讨论

### DB2 和 SQL Server

内联视图 X 将每个姓名中的每个字符都作为 1 行返回。函数 SUBSTR 或 SUBSTRING 从相应的名字中提取出每个字符，然后 ROW\_NUMBER 函数按照字母顺序给每个字符规定序号：

ENAME	C	POS
-----	-	---
ADAMS	A	1
ADAMS	A	2
ADAMS	D	3
ADAMS	M	4
ADAMS	S	5
...		

要将字符串中的每个字母作为 1 行返回，必须遍历此字符串。此项操作使用内联视图 ITER 完成。

现在，每个姓名中的字母已经按照字母顺序排列，最后一步就是将这些字母按照各自的序号重新放回到一起，作为一个字符串。每个字母放置的位置取决于 CASE 语句（行 2～7），如果字符处于某一特定的位置，则将该字符跟下一个求得的值（下一个 CASE 语句）连接起来。因为使用了聚集函数 MAX，每个 POS 位置只返回一个字符，所以每个姓名只返回一行。CASE 表达式计算 6 次，因为这是在表 EMP 中所有名字中最大的字符数。

## MySQL

内联视图 X (行 3~6) 将每个姓名中的每个字符作为一行返回, 函数 SUBSTR 从每个姓名中提取出每个字符:

ENAME	C
-----	-
ADAMS	A
ADAMS	A
ADAMS	D
ADAMS	M
ADAMS	S
...	

内联视图 ITER 用来遍历字符串, 然后, 剩下的工作由 GROUP\_CONCAT 函数来完成。通过指定顺序, 函数除了连接每个字母之外, 还要按照字母的顺序进行排列。

## Oracle

真正的操作由内联视图 X (行 5~11) 来完成, 它将每个姓名中的字符提取出来并且按照字母顺序排列。这是通过遍历字符串, 然后按照这些字符来进行排序而完成的。查询的其余部分用来将姓名重新组合到一起。

单独执行内联视图 X 可以看到被分开的姓名:

OLD_NAME		RN	C
-----	-----	-	-
ADAMS		1	A
ADAMS		2	A
ADAMS		3	D
ADAMS		4	M
ADAMS		5	S
...			

下一步就是按照字母的顺序排列字符, 并且重新构建每个姓名。然后使用函数 SYS\_CONNECT\_BY\_PATH 来将每个字符添加到前一字符的后面:

OLD_NAME	NEW_NAME
-----	-----
ADAMS	A
ADAMS	AA
ADAMS	AAD
ADAMS	AADM
ADAMS	AADMS
...	

最后一步就是保留长度跟用来构建它的姓名长度相同的字符串。

## PostgreSQL

为了增加易读性, 这种解决方案中用视图 V 来遍历字符串。在视图定义中的函数 SUBSTR 用来从每个姓名中提取出每个字符, 故视图返回的结果集为:

ENAME	C
-----	-
ADAMS	A

```
ADAMS  A
ADAMS  D
ADAMS  M
ADAMS  S
...
```

此视图也按照 ENAME 和姓名中的每个字母来排序。内联视图 X (行 15~18) 从视图 V 中返回姓名和各字符, 每个字符在姓名中出现的次数及其所在的位置 (按字母顺序)。

ename	c	cnt	pos
ADAMS	A	2	1
ADAMS	A	2	1
ADAMS	D	1	3
ADAMS	M	1	4
ADAMS	S	1	5

从内联视图 X 返回的 CNT 和 POS 列对此解决方案至关重要。POS 用来排列每个字符, 而 CNT 用来计算字符在每个姓名中出现的次数。最后的步骤就是求出每个字符的位置并且重新构建姓名。需要注意的是每个 CASE 语句实际上有两个 CASE 语句, 这是为了判定某个字符是否在姓名中出现多次, 如果是的话, 就不是直接返回该字符, 而是将 CNT 个该字符连接在一起再返回。聚集函数 MAX 保证每一个姓名只有一行。

## 6.13 判别可作为数值的字符串问题

表中的某列已经定义为字符类型数据, 遗憾的是, 这些行中既有数值数据又有字符数据。考虑视图 V:

```
create view V as
select replace(mixed, ' ', '') as mixed
  from (
select substr(ename,1,2)||
       cast(deptno as char(4))||
       substr(ename,3,2) as mixed
  from emp
 where deptno = 10
 union all
select cast(empno as char(4)) as mixed
  from emp
 where deptno = 20
 union all
select ename as mixed
  from emp
 where deptno = 30
 ) x
```

```
select * from v

MIXED
-----
CL10AR
KI10NG
MI10LL
7369
7566
7788
```

```

7876
7902
ALLEN
WARD
MARTIN
BLAKE
TURNER
JAMES

```

只要返回那些只是数值，或是在其中至少包含一个是数值的行，如果数值与字符混合在一起，需要删除那些字符，只返回数字。例如，从上面的样本数据中要返回如下结果集：

```

MIXED
-----
      10
      10
      10
     7369
     7566
     7788
     7876
     7902

```

## 解决方案

函数 REPLACE 和 TRANSLATE 对于操作字符串和单独的字符非常有用。关键就是先将所有的数字转换成同一个字符，这样就可以很轻易地通过引用一个字符而将数字隔离并识别出来。

### DB2

使用 TRANSLATE、REPLACE 和 POSSTR 函数来隔离每一行中的数字字符。在视图 V 中调用 CAST 是必需的，否则会由于类型转换错误而无法创建视图。由于强制转换为固定长度的 CHAR 类型，所以需要 REPLACE 函数来删除多余的空白。

```

1 select mixed old,
2       cast(
3         case
4           when
5             replace(
6               translate(mixed,'9999999999','0123456789'),'9','') = ''
7           then
8             mixed
9         else replace(
10            translate(mixed,
11              repeat('#',length(mixed)),
12              replace(
13                translate(mixed,'9999999999','0123456789'),'9',''),
14                '#','')
15          end as integer ) mixed
16   from V
17  where posstr(translate(mixed,'9999999999','0123456789'),'9') > 0

```

### MySQL

MySQL 的语法稍有不同，其视图 V 的定义为：

```
create view V as
```

```

select concat(
    substr(ename,1,2),
    replace(cast(deptno as char(4)), ' ', ''),
    substr(ename,3,2)
) as mixed
from emp
where deptno = 10
union all
select replace(cast(empno as char(4)), ' ', '')
from emp where deptno = 20
union all
select ename from emp where deptno = 30

```

因为MySQL不支持TRANSLATE函数，所以必需遍历每行，并且基于一个一个字符地计算值：

```

1 select cast(group_concat(c order by pos separator '') as unsigned)
2     as MIXED1
3   from (
4 select v.mixed, iter.pos, substr(v.mixed,iter.pos,1) as c
5   from V,
6        ( select id pos from t10 ) iter
7  where iter.pos <= length(v.mixed)
8        and ascii(substr(v.mixed,iter.pos,1)) between 48 and 57
9        ) y
10  group by mixed
11  order by 1

```

## Oracle

使用TRANSLATE、REPLACE和POSSTR函数来隔离每一行中的数字字符。视图V中不必调用CAST，由于强制转换为固定长度的CHAR类型，所以需要REPLACE函数来删除多余的空白。如果决定在视图定义中使用显式的类型转换，那么建议转换为VARCHAR2类型：

```

1 select to_number (
2     case
3     when
4         replace(translate(mixed,'0123456789','9999999999'),'9')
5         is not null
6     then
7         replace(
8             translate(mixed,
9                 replace(
10                    translate(mixed,'0123456789','9999999999'),'9'),
11                rpad('#',length(mixed),'#')),'#')
12     else
13         mixed
14     end
15 ) mixed
16 from V
17 where instr(translate(mixed,'0123456789','9999999999'),'9') > 0

```

## PostgreSQL

使用TRANSLATE、REPLACE和STRPOS函数来隔离每一行中的数字字符。视图V中调用CAST不是必需的，由于强制转换为固定长度的CHAR类型，所以需要REPLACE函数来删除多余的空白。如果决定在视图定义中使用显式的类型转换，那么建议转换为VARCHAR类型：

```

1 select cast(
2     case
3     when
4         replace(translate(mixed, '0123456789', '9999999999'), '9', '')
5     is not null
6     then
7         replace(
8             translate(mixed,
9                 replace(
10                    translate(mixed, '0123456789', '9999999999'), '9', ''),
11                    rpad('#', length(mixed), '#')), '#', '')
12        else
13            mixed
14        end as integer ) as mixed
15 from V
16 where strpos(translate(mixed, '0123456789', '9999999999'), '9') > 0

```

## SQL Server

用内置函数 ISNUMERIC 配合使用通配符搜索可以很轻易地识别包含数字的字符串，但是将数字从字符串中提取出来则不容易，因为 SQL Server 不支持 TRANSLATE 函数。

## 讨论

TRANSLATE 函数非常有用，该函数允许用户隔离并识别数字和字符。这里的诀窍就是将所有的数字转换为同一个字符，这样就不必查找不同的数字，而只要查找一个字符。

## DB2、Oracle 和 PostgreSQL

这些 DBMS 所使用的语法互相之间稍微有些不同，但所使用的技术是一样的。在这里将用 PostgreSQL 的解决方案进行讨论。

真正的工作是由 TRANSLATE 和 REPLACE 函数来完成的。要得到最终的结果集需要调用多个函数，每个函数都用在下面的查询列：

```

select mixed as orig,
translate(mixed, '0123456789', '9999999999') as mixed1,
replace(translate(mixed, '0123456789', '9999999999'), '9', '') as mixed2,
translate(mixed,
replace(
translate(mixed, '0123456789', '9999999999'), '9', ''),
rpad('#', length(mixed), '#')) as mixed3,
replace(
translate(mixed,
replace(
translate(mixed, '0123456789', '9999999999'), '9', ''),
rpad('#', length(mixed), '#')), '#', '') as mixed4
from V
where strpos(translate(mixed, '0123456789', '9999999999'), '9') > 0

```

ORIG	MIXED1	MIXED2	MIXED3	MIXED4	MIXED5
CL10AR	CL99AR	CLAR	##10##	10	10
KI10NG	KI99NG	KING	##10##	10	10
MI10LL	MI99LL	MILL	##10##	10	10
7369	9999		7369	7369	7369
7566	9999		7566	7566	7566
7788	9999		7788	7788	7788
7876	9999		7876	7876	7876
7902	9999		7902	7902	7902

首先注意，所有没有包含数字的行全部被删除了，检查一下上面的结果集就会清楚这是如何实现的。在保留的这些行中，有 ORIG 列以及用来组成最终结果集的其他列。提取数值的第一步是用 TRANSLATE 函数，将所有数字转换为 9（可以使用任何数字，9 是任意选择的一个数字），这些值保存在 MIXED1 中。既然所有的数字全部是 9 了，就可以将它们作为一个单独的单位对待。下一步是使用 REPLACE 函数删除所有的数字。因为所有的数字现在都为 9，REPLACE 函数只是查找数字 9，并将其删除即可，这些值被保存在 MIXED2 中。下一步，用 MIXED2 的值求 MIXED3，将 MIXED2 中的值与 ORIG 中的值进行比较，如果 ORIG 中包含 MIXED2 中的任何字符，那么用函数 TRANSLATE 将这些字符转换为“#”。MIXED3 中的结果被显示，所有的字母，而不是数字已经被挑选出来并转换为同一个字符。既然所有的非数字字符全都用 '#' 代表了，就可以将它们作为一个单独的单位对待。接下来求 MIXED4，用 REPLACE 函数在每一行中查找并删除所有的 # 字符，这样就只剩下数字了。最后，将这些数字字符串转换为数值。一步步走过来之后，现在可以清楚 WHERE 子句的机理了。将 MIXED1 的结果传递到 STRPOS，如果其中包含 9（在字符串中第一个 9 的位置），那么其结果必须大于 0。如果某些行的返回值大于 0，就表示在此行中至少有一个数字，需要保留。

## MySQL

第一步就是遍历每个字符串，检查每个字符并且判断该字符是否为数字：

```
select v.mixed, iter.pos, substr(v.mixed,iter.pos,1) as c
  from V,
       ( select id pos from t10 ) iter
 where iter.pos <= length(v.mixed)
 order by 1,2
```

mixed	pos	c
7369	1	7
7369	2	3
7369	3	6
7369	4	9
...		
ALLEN	1	A
ALLEN	2	L
ALLEN	3	L
ALLEN	4	E
ALLEN	5	N
...		
CL10AR	1	C
CL10AR	2	L
CL10AR	3	1
CL10AR	4	0
CL10AR	5	A
CL10AR	6	R

既然可以分离字符串中的各个字符，下一步就是保留那些在 C 列中是数字的行。

```
select v.mixed, iter.pos, substr(v.mixed,iter.pos,1) as c
  from V,
       ( select id pos from t10 ) iter
```



```

where iter.pos <= length(v.mixed)
and ascii(substr(v.mixed,iter.pos,1)) between 48 and 57
order by 1,2

```

mixed	pos	c
7369	1	7
7369	2	3
7369	3	6
7369	4	9
...		
CL10AR	3	1
CL10AR	4	0
...		

此时，所有行的 C 列中都为数字，下一步是用 GROUP\_CONCAT 来将数字连接起来构成完整的数并存于 MIXED 中，然后将它们转换成数值作为最终结果：

```

select cast(group_concat(c order by pos separator '') as unsigned)
as MIXED1
from (
select v.mixed, iter.pos, substr(v.mixed,iter.pos,1) as c
from V,
( select id pos from t10 ) iter
where iter.pos <= length(v.mixed)
and ascii(substr(x.mixed,iter.pos,1)) between 48 and 57
) y
group by mixed
order by 1

```

MIXED1
10
10
10
7369
7566
7788
7876
7902

最后要注意的是，一定要记住在每个字符串中的所有数字都连接到一起，形成一个数值。例如，如果输入值为“99Gennick87”其结果为 9987。尤其是在处理序列化的数据时，一定要考虑这个问题。

## 6.14 提取第 n 个分隔的子串

### 问题

从字符串中提取出一个指定的、由分隔符隔开的子字符串。考虑下面的视图 V，该视图用来生成此问题的源数据。

```

create view V as
select 'mo,larry,curly' as name
from t1
union all
select 'tina,gina,jaunita,regina,leena' as name

```

```
from t1
```

此视图输出值如下所列：

```
select * from v
NAME
-----
mo,larry,curly
tina,gina,jaunita,regina,leena
```

要提取出每行中第二个姓名，最终的结果集应当如下所示：

```
SUB
----
larry
gina
```

## 解决方案

解决此问题的关键就是将每个姓名作为单独的行返回，并保留每个姓名在列表中的顺序。如何解决该问题取决于所使用的 DBMS。

### DB2

在遍历由视图 V 返回的 NAME 字段后，使用函数 ROW\_NUMBER 只保留每个字符串中的第二个姓名：

```
1 select substr(c,2,locate(',',c,2)-2)
2   from (
3 select pos, name, substr(name, pos) c,
4        row_number( ) over(partition by name
5                             order by length(substr(name,pos)) desc) rn
6   from (
7 select ',' || csv.name || ',' as name,
8        cast(iter.pos as integer) as pos
9   from V csv,
10        (select row_number( ) over( ) pos from t100 ) iter
11  where iter.pos <= length(csv.name)+2
12        ) x
13  where length(substr(name,pos)) > 1
14        and substr(substr(name,pos),1,1) = ','
15        ) y
16  where rn = 2
```

### MySQL

在遍历由视图 V 返回的 NAME 字段后，根据逗号的位置来只返回每个字符串中的第二个姓名：

```
1 select name
2   from (
3 select iter.pos,
4        substring_index(
5          substring_index(src.name,',',iter.pos),',',-1) name
6   from V src,
7        (select id pos from t10) iter,
8  where iter.pos <=
9        length(src.name)-length(replace(src.name,',',''))
10        ) x
11  where pos = 2
```

## Oracle

在遍历由视图 V 返回的 NAME 字段后，用 SUBSTR 和 INSTR 函数来找回每个列表中的第二个姓名。

```

1 select sub
2   from (
3 select iter.pos,
4        src.name,
5        substr( src.name,
6               instr( src.name,',',1,iter.pos )+1,
7               instr( src.name,',',1,iter.pos+1 ) -
8               instr( src.name,',',1,iter.pos )-1) sub
9   from (select ',||name||',' as name from V) src,
10        (select rownum pos from emp) iter
11  where iter.pos < length(src.name)-length(replace(src.name',''))
12        )
13  where pos = 2

```

## PostgreSQL

使用 SPLIT\_PART 函数将每个单独的名字作为一行返回：

```

1 select name
2   from (
3 select iter.pos, split_part(src.name,',',iter.pos) as name
4   from (select id as pos from t10) iter,
5        (select cast(name as text) as name from v) src
7  where iter.pos <=
8         length(src.name)-length(replace(src.name',''))+1
9         ) x
10  where pos = 2

```

## SQL Server

在遍历由视图 V 返回的 NAME 字段后，使用函数 ROW\_NUMBER 来保留每个字符串中的第二个姓名。

```

1 select substring(c,2,charindex(',',c,2)-2)
2   from (
3 select pos, name, substring(name, pos, len(name)) as c,
4        row_number( ) over(
5          partition by name
6          order by len(substring(name,pos,len(name))) desc) rn
7   from (
8 select ',' + csv.name + ',' as name,
9        iter.pos
10  from V csv,
11       (select id as pos from t100 ) iter
12  where iter.pos <= len(csv.name)+2
13        ) x
14  where len(substring(name,pos,len(name))) > 1
15        and substring(substring(name,pos,len(name)),1,1) = ','
16        ) y
17  where rn = 2

```

## 讨论

### DB2 和 SQL Server

这两种 DBMS 解决本问题的方法是相同的，只是语法稍有差别。这里用 DB2 的解决方案来进行讨论。用内联视图 X 表示遍历字符串的结果：

```
select '||csv.name||',' as name,
       iter.pos
  from v csv,
       (select row_number( ) over( ) pos from t100 ) iter
 where iter.pos <= length(csv.name)+2
```

EMPS	POS
,tina,gina,jaunita,regina,leena,	1
,tina,gina,jaunita,regina,leena,	2
,tina,gina,jaunita,regina,leena,	3
...	

下一步是逐个“穿越”每个字符串中的每个字符：

```
select pos, name, substr(name, pos) c,
       row_number( ) over(partition by name
                           order by length(substr(name, pos)) desc) rn
  from (
select '||csv.name||',' as name,
       cast(iter.pos as integer) as pos
  from v csv,
       (select row_number( ) over( ) pos from t100 ) iter
 where iter.pos <= length(csv.name)+2
       ) x
 where length(substr(name,pos)) > 1
```

POS	EMPS	C	RN
1	,mo,larry,curly,	,mo,larry,curly,	1
2	,mo,larry,curly,	mo,larry,curly,	2
3	,mo,larry,curly,	o,larry,curly,	3
4	,mo,larry,curly,	,larry,curly,	4
...			

将字符串分解为不同的部分之后，只需要判别出需要保留哪些行就可以了。所需要的行由逗号开始，其余的行都可以删除：

```
select pos, name, substr(name,pos) c,
       row_number( ) over(partition by name
                           order by length(substr(name, pos)) desc) rn
  from (
select '||csv.name||',' as name,
       cast(iter.pos as integer) as pos
  from v csv,
       (select row_number( ) over( ) pos from t100 ) iter
 where iter.pos <= length(csv.name)+2
       ) x
 where length(substr(name,pos)) > 1
       and substr(substr(name,pos),1,1) = ','
```

POS	EMPS	C	RN
1	,mo,larry,curly,	,mo,larry,curly,	1
4	,mo,larry,curly,	,larry,curly,	2
10	,mo,larry,curly,	,curly,	3
1	,tina,gina,jaunita,regina,leena,	,tina,gina,jaunita,regina,leena,	1
6	,tina,gina,jaunita,regina,leena,	,gina,jaunita,regina,leena,	2
11	,tina,gina,jaunita,regina,leena,	,jaunita,regina,leena,	3
19	,tina,gina,jaunita,regina,leena,	,regina,leena,	4
26	,tina,gina,jaunita,regina,leena,	,leena,	5

这一步骤非常重要，因为它决定了如何得到第n个子串。注意，在此查询中很多行被去掉了，全是因为如下的条件语句：

```
substr(substr(name,pos),1,1) = ','
```

可以注意到，本来在队列中排在第四的“larry,curly,”，现在在第二位。记住，WHERE子句在SELECT之前执行，所以，先保留由逗号开始的行，然后由ROW\_NUMBER函数确定序号。到这里就很清楚了，要得到第n个子串，只要返回RN等于n的行即可。最后一步就是保留需要的行（这里是RN等于2的行），并且使用SUBSTR函数来从行中提取出姓名。所要保留的是每行的第一个名字，从“larry,curly,”行中是“larry”，而从“gina,jaunita,regina,leena,”行中是“gina”。

## MySQL

内联视图X遍历每个字符串，可以通过计算在字符串中分隔符的数量，来判定每个字符串中有多少值。

```
select iter.pos, src.name
  from (select id pos from t10) iter,
       v src
 where iter.pos <=
       length(src.name)-length(replace(src.name,',',''))
```

pos	name
1	mo,larry,curly
2	mo,larry,curly
1	tina,gina,jaunita,regina,leena
2	tina,gina,jaunita,regina,leena
3	tina,gina,jaunita,regina,leena
4	tina,gina,jaunita,regina,leena

在这种情况下，在每个字符串相应的行数比其中的项数小1，因为就需要这么多。函数SUBSTRING\_INDEX负责提取出所需要的值：

```
select iter.pos,src.name name1,
       substring_index(src.name,',',iter.pos) name2,
       substring_index(
         substring_index(src.name,',',iter.pos),',',-1) name3
  from (select id pos from t10) iter,
       v src
 where iter.pos <=
       length(src.name)-length(replace(src.name,',',''))
```

pos	name1	name2	name3
1	mo,larry,curly	mo	mo
2	mo,larry,curly	mo,larry	larry
1	tina,gina,jaunita,regina,leena	tina	tina
2	tina,gina,jaunita,regina,leena	tina,gina	gina
3	tina,gina,jaunita,regina,leena	tina,gina,jaunita	jaunita
4	tina,gina,jaunita,regina,leena	tina,gina,jaunita,regina	regina

这里显示了3个姓名字段，这样读者可以看出嵌套调用SUBSTRING\_INDEX函数的工作机理：里层的调用返回在第n个逗号左侧的所有字符；外层调用返回最后一个逗号右面的所有内容。最后一步就是保留POS等于n（本例为2）各行的NAME3字段。

## Oracle

内联视图遍历每个字符串，返回每个字符串的次数取决于每个字符串中的项数。解决方案中是通过计算字符串中分隔符的数量来判定其项数的。因为每个字符串是由逗号括起来的，字符串中的项数就是逗号的数量减1。然后，用这些字符串的“并集”跟一个基数不小于这些串中最大项数的表做联接。函数SUBSTR和INSTR用POS值来分解每个字符串。

```
select iter.pos, src.name,
       substr( src.name,
              instr( src.name, ',', 1, iter.pos )+1,
              instr( src.name, ',', 1, iter.pos+1 ) -
              instr( src.name, ',', 1, iter.pos )-1) sub
  from (select '||name||' as name from v) src,
       (select rownum pos from emp) iter
 where iter.pos < length(src.name)-length(replace(src.name, ','))
```

POS	NAME	SUB
1	,mo,larry,curly,	mo
1	, tina,gina,jaunita,regina,leena,	tina
2	,mo,larry,curly,	larry
2	, tina,gina,jaunita,regina,leena,	gina
3	,mo,larry,curly,	curly
3	, tina,gina,jaunita,regina,leena,	jaunita
4	, tina,gina,jaunita,regina,leena,	regina
5	, tina,gina,jaunita,regina,leena,	leena

在SUBSTR中对INSTR函数的第一次调用是为了确定要提取的子串的开始位置；接下来的调用是分别找出第n个逗号的位置（与开始位置相同）和第n+1个逗号的位置，将这两个值相减就是要提取出来的子串的长度。因为每个值都被分解成单独的一行，所以只需要用WHERE POS=n就可以保留第n个字符串（本例用了where POS=2，表示第二个子串）。

## PostgreSQL

内联视图X遍历每个字符串，在每个字符串中有多少值，决定了该视图返回多少行，在每个字符串中分隔符的数量加1就是每个字符串中的项数。函数SPLIT\_PART用POS的值来查找第n个分隔符，并且将字符串分解为值项：

```
select iter.pos, src.name as name1,
       split_part(src.name, ',', iter.pos) as name2
  from (select id as pos from t10) iter,
       (select cast(name as text) as name from v) src
 where iter.pos <=
       length(src.name)-length(replace(src.name, ','))+1
```

pos	name1	name2
1	mo,larry,curly	mo
2	mo,larry,curly	larry
3	mo,larry,curly	curly
1	tina,gina,jaunita,regina,leena	tina
2	tina,gina,jaunita,regina,leena	gina
3	tina,gina,jaunita,regina,leena	jaunita
4	tina,gina,jaunita,regina,leena	regina
5	tina,gina,jaunita,regina,leena	leena

在这里显示了NAME字段两次，以便于看出 SPLIT\_PARTPOS 函数是如何用POS值分解每个字符串的。一旦每个字符串被分解，最后一步就是保留POS等于n（本例中POS等于2）的行，从而得到所要求的第n个子串。

## 6.15 分解IP地址

### 问题

将一个IP地址字段分解到列中，考虑下面列出的IP地址：

111.22.3.4

要得到如下所示的查询结果：

A	B	C	D
111	22	3	4

### 解决方案

此问题的解决方案取决于所使用的DBMS所提供的内部函数。不管哪种DBMS，关键就是要能够找出点的位置及它们前后的数字。

#### DB2

用递归的WITH字句来模拟对IP地址的反复处理，而用SUBSTR函数将其分解。在IP地址最前面增加一个句点，使每组数字前面都有句点，从而可以用同样的方法处理：

```

1 with x (pos,ip) as (
2   values (1, '.92.111.0.222')
3   union all
4   select pos+1, ip from x where pos+1 <= 20
5 )
6 select max(case when rn=1 then e end) a,
7        max(case when rn=2 then e end) b,
8        max(case when rn=3 then e end) c,
9        max(case when rn=4 then e end) d
10  from (
11 select pos, c, d,
12        case when posstr(d, '.') > 0 then substr(d, 1, posstr(d, '.')-1)
13             else d
14        end as e,
15        row_number( ) over(order by pos desc) rn
16  from (
17 select pos, ip, right(ip, pos) as c, substr(right(ip, pos), 2) as d
18  from x
19  where pos <= length(ip)
20        and substr(right(ip, pos), 1, 1) = '.'
21        ) x
22        ) y

```

#### MySQL

使用SUBSTR\_INDEX函数可以很轻松地分解IP地址：

```

1 select substring_index(substring_index(y.ip, '.', 1), '.', -1) a,
2        substring_index(substring_index(y.ip, '.', 2), '.', -1) b,

```

```

3      substring_index(substring_index(y.ip,'.',3),'',-1) c,
4      substring_index(substring_index(y.ip,'.',4),'',-1) d
5  from (select '92.111.0.2' as ip from t1) y

```

## Oracle

使用内置的函数 SUBSTR 和 INSTR 来分解 IP 地址：

```

1 select ip,
2      substr(ip, 1, instr(ip,'.')-1 ) a,
3      substr(ip, instr(ip,'.')+1,
4             instr(ip,'.',1,2)-instr(ip,'.')-1 ) b,
5      substr(ip, instr(ip,'.',1,2)+1,
6             instr(ip,'.',1,3)-instr(ip,'.',1,2)-1 ) c,
7      substr(ip, instr(ip,'.',1,3)+1 ) d
8  from (select '92.111.0.2' as ip from t1)

```

## PostgreSQL

使用内置函数 SPLIT\_PART 来分解 IP 地址：

```

1 select split_part(y.ip,'.',1) as a,
2      split_part(y.ip,'.',2) as b,
3      split_part(y.ip,'.',3) as c,
4      split_part(y.ip,'.',4) as d
5  from (select cast('92.111.0.2' as text) as ip from t1) as y

```

## SQL Server

用递归的 WITH 字句来模拟对 IP 地址的反复处理，而用 SUBSTR 函数将其分解。在 IP 地址最前面增加一个句点，使每组数字前面都有句点，从而可以用同样的方法处理：

```

1  with x (pos,ip) as (
2      select 1 as pos,'.92.111.0.222' as ip from t1
3      union all
4      select pos+1,ip from x where pos+1 <= 20
5  )
6  select max(case when rn=1 then e end) a,
7         max(case when rn=2 then e end) b,
8         max(case when rn=3 then e end) c,
9         max(case when rn=4 then e end) d
10 from (
11 select pos,c,d,
12        case when charindex('.',d) > 0
13             then substring(d,1,charindex('.',d)-1)
14             else d
15        end as e,
16        row_number( ) over(order by pos desc) rn
17 from (
18 select pos, ip,right(ip,pos) as c,
19        substring(right(ip,pos),2,len(ip)) as d
20 from x
21 where pos <= len(ip)
22       and substring(right(ip,pos),1,1) = '.'
23       ) x
24       ) y

```

## 讨论

用数据库的内置函数，可以很轻松地遍历字符串的每一部分。关键就是要能够定位每个地址中句点的位置，然后就可以分解出句点之间的数值。



# 使用数字

本章将介绍有关数字的常用操作，包括数字运算。虽然一般认为对于复杂计算，SQL 并非首选；然而对于日常的数字运算，它还是非常有效的。

**注意：**本章使用了聚集函数和 GROUP BY 子句。如果对分组的有关概念不太熟悉，请阅读“附录 A”中的第一部分“分组”。

## 7.1 计算平均值

### 问题

计算某个列的平均值，它可以包含表中的所有行，也可以只包含其中的某个子集。例如，计算所有职员的平均工资以及每个部门的平均工资。

### 解决方案

当计算所有职员的平均工资时，只需把 AVG 函数应用于工资列即可。不带 WHERE 子句时，会对所有非 NULL 工资值计算平均值：

```
1 select avg(sal) as avg_sal
2   from emp
```

```
      AVG_SAL
-----
2073.21429
```

要计算每个部门的平均工资，需使用 GROUP BY 子句，它按每个部门创建分组：

```
1 select deptno, avg(sal) as avg_sal
2   from emp
3  group by deptno
```

```
      DEPTNO      AVG_SAL
-----
10 2916.66667
20  2175
30 1566.66667
```

## 讨论

如果以整个表作为一个组或一个窗口计算平均值，则只需对相应列使用 AVG 函数，而不要使用 GROUP BY 子句。请注意，AVG 函数会忽略 NULL 值。下面给出了忽略 NULL 值的效果：

```
create table t2(sal integer)
insert into t2 values (10)
insert into t2 values (20)
insert into t2 values (null)

select avg(sal)      select distinct 30/2
  from t2           from t2

  AVG(SAL)           30/2
-----
          15        15

select avg(coalesce(sal,0))  select distinct 30/3
  from t2                   from t2

AVG(COALESCE(SAL,0))        30/3
-----
                10        10
```

COALESCE 函数会返回参数值列表中的第一个非 NULL 值。如果把 SAL 值中的 NULL 转换为 0，平均值就会改变。当调用集合函数时，一定要想好如何处理 NULL 值。

该解决方案的第二部分是使用 GROUP BY（第三行），根据职员的关系，把职员记录分成组。GROUP BY 会自动让集合函数（如 AVG）为每个组返回一个结果。在这个例子中，AVG 把每个部门的职员记录当作一个组，从而执行一次。

顺便提一下，GROUP BY 依据的列不一定要包含在 SELECT 列表中。例如：

```
select avg(sal)
  from emp
  group by deptno

  AVG(SAL)
-----
2916.66667
      2175
1566.66667
```

即使在 SELECT 子句中不包含 DEPTNO，也可以按它分组。在 SELECT 子句中引入要分组的列通常会增加可读性，但这并不是强制性的。然而，一定不能把不在 GROUP BY 子句中的列放在 SELECT 列表中。

## 参阅

有关 GROUP BY 功能的资料，请参阅“附录 A”。

## 7.2 求某列中的最小 / 最大值

### 问题

计算给定列中的最大值和最小值。例如，计算所有职员的最大工资和最低工资，以及每个部门的最大工资和最低工资。

### 解决方案

要查所有职员的最低工资和最高工资，只需分别使用函数 MIN 和 MAX：

```
1 select min(sal) as min_sal, max(sal) as max_sal
2   from emp
```

MIN_SAL	MAX_SAL
800	5000

要查每个部门的最低工资和最高工资，在使用函数 MIN 和 MAX 的同时，还要使用 GROUP BY 子句：

```
1 select deptno, min(sal) as min_sal, max(sal) as max_sal
2   from emp
3  group by deptno
```

DEPTNO	MIN_SAL	MAX_SAL
10	1300	5000
20	800	3000
30	950	2850

### 讨论

如果将整个表作为一个组或一个窗口查最大值或最小值，那么只需针对相应的列使用 MIN 或 MAX 函数，而不要使用 GROUP BY 子句。

请注意，MIN 和 MAX 函数会忽略 NULL 值，而且允许包含 NULL 组，组中的列也允许 NULL 值。下面这些例子的目的是用一个使用 GROUP BY 字句的查询，得到两个包含 NULL 值的组（DEPTNO 10 和 30）：

```
select deptno, comm
  from emp
 where deptno in (10,30)
 order by 1
```

DEPTNO	COMM
10	
10	
10	
30	300
30	500
30	
30	0
30	1300
30	

```
select min(comm), max(comm)
  from emp
```

```

      MIN(COMM)    MAX(COMM)
-----
              0      1300

select deptno, min(comm), max(comm)
   from emp
  group by deptno

      DEPTNO    MIN(COMM)    MAX(COMM)
-----
          10
          20
          30              0      1300

```

“附录 A”将提到，即使 SELECT 子句中仅包含聚集函数，也可以按表中的其他列分组，例如：

```

select min(comm), max(comm)
   from emp
  group by deptno

      MIN(COMM)    MAX(COMM)
-----
              0      1300

```

这里是按 DEPTNO 进行分组的，而它并没有出现在 SELECT 子句中。在 SELECT 子句中引入依据其分组的列通常会提高可读性，但这并不是强制性的。然而，对于 GROUP BY 查询的 SELECT 列表中的列，一定要在 GROUP BY 子句中列出来。

## 参阅

有关 GROUP BY 功能的资料，请参阅“附录 A”。

## 7.3 对某列的值求和

### 问题

计算某个列中所有值的和，例如，计算所有职员的工资总额。

### 解决方案

如果将整个表作为一个组或一个窗口求和，则只需对相应列使用 SUM 函数，而不要使用 GROUP BY 子句：

```

1 select sum(sal)
2   from emp

      SUM(SAL)
-----
      29025

```

如果创建了多个数据组或多个窗口，则使用 SUM 函数的同时，还要使用 GROUP BY 子句。下面的例子将按部门计算职员的工资总额：

```

1 select deptno, sum(sal) as total_for_dept
2   from emp

```

### 3 group by deptno

DEPTNO	TOTAL_FOR_DEPT
10	8750
20	10875
30	9400

## 讨论

要查找每个部门的所有工资总额，需创建数据组或“窗口”。把一个部门每个职员的工资都加起来，就会得到该部门的工资总额。这是 SQL 中有关“聚集”的一个例子，因为这里关注的不是诸如每个职员的工资之类的细节信息，而是每个部门总的结果。请注意，SUM 函数会忽略 NULL，但可以存在 NULL 组，下面会看到 DEPTNO 10 中的职员都没有佣金，在按 DEPTNO 10 分组对 COMM 中值求和时，就得到一个组，SUM 函数返回的值是 NULL：

```
select deptno, comm
  from emp
 where deptno in (10,30)
 order by 1
```

DEPTNO	COMM
10	
10	
10	
30	300
30	500
30	
30	0
30	1300
30	

```
select sum(comm)
  from emp
```

SUM(COMM)
2100

```
select deptno, sum(comm)
  from emp
 where deptno in (10,30)
 group by deptno
```

DEPTNO	SUM(COMM)
10	
30	2100

## 参阅

有关 GROUP BY 功能的资料，请参阅“附录 A”。

## 7.4 求一个表的行数

### 问题

计算一个表的行数，或计算某个列中值的个数。例如，找到职员总数以及每个部门的职员数。

## 解决方案

如果以整个表作为一个组或一个窗口计算行数，则只需使用 COUNT 函数及 “\*” 字符：

```
1 select count(*)
2   from emp

COUNT(*)
-----
14
```

如果要创建多个数据组或窗口，则使用 COUNT 函数的同时，还要使用 GROUP BY 子句：

```
1 select deptno, count(*)
2   from emp
3  group by deptno

DEPTNO    COUNT(*)
-----
10         3
20         5
30         6
```

## 讨论

要计算每个部门的职员总数，需创建数据组或“窗口”。每找到一个职员就给该职员对应部门的职员数加1，这样就会生成该部门的职员总数。这是 SQL 中有关“聚集”的一个例子，因为这里关注的不是诸如每个职员的工资之类的细节信息，而是每个部门总的结果。请注意，当把列名作为参数传递给 COUNT 函数时，就会忽略 NULL；但如果给它传递 “\*” 字符或常量，则会包含 NULL。请看下面的例子：

```
select deptno, comm
  from emp

DEPTNO    COMM
-----
20
30        300
30        500
20
30       1300
30
10
20
10
30         0
20
30
20
10

select count(*), count(deptno), count(comm), count('hello')
  from emp

COUNT(*) COUNT(DEPTNO) COUNT(COMM) COUNT('HELLO')
-----
14         14         4         14

select deptno, count(*), count(comm), count('hello')
```

```

      from emp
      group by deptno

```

DEPTNO	COUNT(*)	COUNT(COMM)	COUNT('HELLO')
10	3	0	3
20	5	0	5
30	6	4	6

对于传递给 COUNT 函数的列，如果所有行都是空的，或如果表为空，那么 COUNT 会返回 0。也应该注意，即使 SELECT 子句中仅包括聚集函数，也可以按表中的其他列进行分组，例如：

```

select count(*)
  from emp
  group by deptno

```

COUNT(*)
3
5
6

注意，这里是按 DEPTNO 进行分组的，而它并没有出现在 SELECT 子句中。在 SELECT 子句中引入依据其分组的列通常会提高可读性，但这并不是强制性的。如果在 SELECT 列表中真的引入了它，那么一定要在 GROUP BY 子句中把它列出来。

## 参阅

有关 GROUP BY 功能的资料，请参阅“附录 A”。

## 7.5 求某列值的个数

### 问题

计算某个列中非 NULL 值的个数。例如，找出拥有佣金的职员数。

### 解决方案

计算 EMP 表中 COMM 列中非 NULL 值的个数：

```

select count(comm)
  from emp
  COUNT(COMM)

```

4
---

### 讨论

当采用 COUNT(\*) 格式时，实际上是数行数（因而实际值是 NULL 和非 NULL 的行都包括了）。但如果要对某个列进行 COUNT 运算，则要计算的是该列中具有非 NULL 值的行数。上一节已经讨论过这种差别。在该解决方案中，COUNT(COMM) 返回了 COMM 列中的非 NULL 值的个数。由于只有被任命的职员才有佣金，所以 COUNT(COMM) 的结果是这些被任命职员的总数。

## 7.6 生成累计和

### 问题

计算某个列中所有值的累计和。

### 解决方案

下面给出了一种解决方案，它展示了如何计算所有职员工资的累计和。为增加可读性，其结果是按 SAL 排序的，这样就能够很容易地观察到累计和变化的过程。

#### DB2 和 Oracle

使用窗口版本的 SUM 函数计算累计和：

```
1 select ename, sal,
2        sum(sal) over (order by sal, empno) as running_total
3   from emp
4  order by 2
```

ENAME	SAL	RUNNING_TOTAL
SMITH	800	800
JAMES	950	1750
ADAMS	1100	2850
WARD	1250	4100
MARTIN	1250	5350
MILLER	1300	6650
TURNER	1500	8150
ALLEN	1600	9750
CLARK	2450	12200
BLAKE	2850	15050
JONES	2975	18025
SCOTT	3000	21025
FORD	3000	24025
KING	5000	29025

#### MySQL、PostgreSQL 和 SQL Server

使用标量子查询计算累计和（由于不使用 SUM OVER 这类窗口函数，因此就不能像在 DB2 和 Oracle 解决方案中那样容易地按 SAL 给结果排序）。不管怎么说，累计和是正确的（最终结果与上一节相同），但由于没有进行排序，其中间值有所不同：

```
1 select e.ename, e.sal,
2        (select sum(d.sal) from emp d
3         where d.empno <= e.empno) as running_total
4   from emp e
5  order by 3
```

ENAME	SAL	RUNNING_TOTAL
SMITH	800	800
ALLEN	1600	2400
WARD	1250	3650
JONES	2975	6625
MARTIN	1250	7875
BLAKE	2850	10725
CLARK	2450	13175
SCOTT	3000	16175



KING	5000	21175
TURNER	1500	22675
ADAMS	1100	23775
JAMES	950	24725
FORD	3000	27725
MILLER	1300	29025

## 讨论

生成累计和是因使用新的ANSI窗口函数而得以简化的任务之一。对于不支持这些窗口函数的DBMS，需要使用标量子查询（按取值唯一的字段联接）。

## DB2 和 Oracle

窗口函数SUM OVER能够非常容易地生成累计和。该解决方案中的ORDER BY子句不仅包含SAL列，而且还包含EMPNO列（主键），以避免累计和中出现重复值。下面例子中的RUNNING\_TOTAL2列示意了存在重复值时可能带来的问题：

```
select empno, sal,
       sum(sal)over(order by sal,empno) as running_total1,
       sum(sal)over(order by sal) as running_total2
  from emp
 order by 2
```

ENAME	SAL	RUNNING_TOTAL1	RUNNING_TOTAL2
SMITH	800	800	800
JAMES	950	1750	1750
ADAMS	1100	2850	2850
WARD	1250	4100	5350
MARTIN	1250	5350	5350
MILLER	1300	6650	6650
TURNER	1500	8150	8150
ALLEN	1600	9750	9750
CLARK	2450	12200	12200
BLAKE	2850	15050	15050
JONES	2975	18025	18025
SCOTT	3000	21025	24025
FORD	3000	24025	24025
KING	5000	29025	29025

对于WARD、MARTIN、SCOTT和FORD，RUNNING\_TOTAL2中的值都不正确。他们的工资分别出现了多次，这些重复值都被加在一起计入累计和。这就是需要使用EMPNO（它是唯一的）才能生成与RUNNING\_TOTAL1一样的（正确）结果的原因。大家想一想：对于ADAMS，RUNNING\_TOTAL1的值为2850，RUNNING\_TOTAL2把WARD的工资1250与2850相加，应该得到4100，然而，RUNNING\_TOTAL2却返回了5350，这是为什么呢？因为WARD和MARTIN的SAL相同，他们两个的工资（1250）加在一起就等于2500，然后再加2850，就得到5350。如果指定按不会有重复值的列组合（例如，SAL和EMPNO的取值组合都是唯一的）排序，就能确保生成正确的累计和。

## MySQL、PostgreSQL 和 SQL Server

在这些DBMS完全支持窗口函数之前，可以使用标量子查询计算累计和。一定要按取值唯一的列联接，否则一旦存在像工资重复这样的情况，就会产生不正确的累计和。本节

解决方案的关键是把 D.EMPNO 与 E.EMPNO 联接起来，它会返回（求和）每个满足 D.EMPNO 小于或等于 E.EMPNO D.SAL。为了更容易理解这些内容，可以重新编写标量子查询，把它写成职员之间的联接：

```
select e.ename as ename1, e.empno as empno1, e.sal as sal1,
       d.ename as ename2, d.empno as empno2, d.sal as sal2
  from emp e, emp d
 where d.empno <= e.empno
        and e.empno = 7566
```

ENAME	EMPNO1	SAL1	ENAME	EMPNO2	SAL2
JONES	7566	2975	SMITH	7369	800
JONES	7566	2975	ALLEN	7499	1600
JONES	7566	2975	WARD	7521	1250
JONES	7566	2975	JONES	7566	2975

EMPNO2 中的每个值与 EMPNO1 中的每个值相比较。对于 EMPNO2 值小于等于 EMPNO1 值的所有行，都会把 SAL2 值加入总和。在这个例子中，职员 Smith、Allen、Ward 和 Jones 的 EMPNO 值都与 Jones 的 EMPNO 值相比较。由于这四个职员的 EMPNO 都满足小于等于 Jones 的 EMPNO 的条件，所以会把这些工资加起来，而那些大于 Jones 的 EMPNO 的职员都不会计入 SUM 中。完整的查询的计算方法是：将所有 EMPNO 小于等于 7934（Miller 的 EMPNO，这个表中的最大值）的所有职员的工资加起来。

## 7.7 生成累乘积

### 问题

计算某个数字列的累乘积。其操作方式与“计算累计和”相似，只是使用乘法而不是加法。

### 解决方案

作为例子，本解决方案中都计算职员工资的累乘积。虽然工资的累乘积没有多大用处，然而可以很容易地把该技巧用于其他更有用的领域。

#### DB2 和 Oracle

使用窗口函数 SUM OVER，用对数相加来模拟乘法操作：

```
1 select empno,ename,sal,
2       exp(sum(ln(sal))over(order by sal,empno)) as running_prod
3   from emp
4  where deptno = 10
```

EMPNO	ENAME	SAL	RUNNING_PROD
7934	MILLER	1300	1300
7782	CLARK	2450	3185000
7839	KING	5000	15925000000

在 SQL 中，对小于等于 0 的值取对数是无效的。如果表中包含这样的值，一定要避免把这些无效的值传递给 SQL 的 LN 函数。为了增加可读性，该解决方案并没有对无效值和

NULL 值采取防范措施，但自己编写代码时，一定要考虑是否需要这种预防。如果一定要用到负值和 0 值，那么这种解决方案不合适。

Oracle 独有的另一种解决方案是使用 Oracle Database 10g 新引入的 MODEL 子句。在下面的例子中，每个 SAL 都是负数，这表明累乘积允许出现负值：

```

1 select empno, ename, sal, tmp as running_prod
2   from (
3   select empno,ename,-sal as sal
4     from emp
5    where deptno=10
6   )
7 model
8   dimension by(row_number()over(order by sal desc) rn )
9   measures(sal, 0 tmp, empno, ename)
10  rules (
11    tmp[any] = case when sal[cv()-1] is null then sal[cv()]
12                  else tmp[cv()-1]*sal[cv()]
13                end
14 )

```

EMPNO	ENAME	SAL	RUNNING_PROD
7934	MILLER	-1300	-1300
7782	CLARK	-2450	3185000
7839	KING	-5000	-15925000000

## MySQL、PostgreSQL 和 SQL Server

还可以使用对数相加的方法，但这些平台并不支持窗口函数，因此用标量子查询取而代之：

```

1 select e.empno,e.ename,e.sal,
2        (select exp(sum(ln(d.sal)))
3         from emp d
4         where d.empno <= e.empno
5               and e.deptno=d.deptno) as running_prod
6   from emp e
7  where e.deptno=10

```

EMPNO	ENAME	SAL	RUNNING_PROD
7782	CLARK	2450	2450
7839	KING	5000	12250000
7934	MILLER	1300	15925000000

SQL Server 用户使用 LOG 代替 LN。

## 讨论

除了 MODEL 子句方案（仅对 Oracle Database 10g 或更高版本可用）之外，所有解决方案都利用了乘法运算的特性，按下列步骤用加法进行计算：

1. 计算各自的自然对数
2. 计算这些对数的和
3. 对结果进行数学常量 e 的幂运算（使用 EXP 函数）

当采用这种方法时，需要注意，对于0值和负值，这种方法不可行，因为任何小于等于0的值都超出了SQL对数的定义域。

## DB2 和 Oracle

有关窗口函数SUM OVER的功能，请参阅“生成累计和”一节。

对于Oracle Database 10g或更高版本，可以使用MODEL子句生成累乘积。同时使用MODEL子句及窗口函数ROW\_NUMBER，很容易就能访问前面的行。可以像访问数组一样访问MEASURES列表中的每一项。然后，可以使用DIMENSIONS列表中的项（由ROW\_NUMBER返回的值，别名RN）搜索该数组：

```
select empno, ename, sal, tmp as running_prod, rn
  from (
select empno, ename, -sal as sal
  from emp
 where deptno=10
    )
 model
  dimension by(row_number()over(order by sal desc) rn )
 measures(sal, 0 tmp, empno, ename)
 rules ()
```

EMPNO	ENAME	SAL	RUNNING_PROD	RN
7934	MILLER	-1300	0	1
7782	CLARK	-2450	0	2
7839	KING	-5000	0	3

观察一下，会发现SAL[1]的值为-1300。由于数字逐一连续递增、没有间隙，所以可以通过减1来引用前一行。RULES子句如下：

```
rules (
  tmp[any] = case when sal[cv()-1] is null then sal[cv()]
                  else tmp[cv()-1]*sal[cv()]
            end
)
```

它使用内置操作符ANY处理每一行，而并未进行硬编码。这个例子中ANY的值分别为1、2和3。把TMP[n]初始化为0。通过计算相应SAL行的当前值（函数CV返回当前值），可以给TMP[n]指定一个值。把TMP[1]初始化为0，把SAL[1]初始化为-1300。SAL[0]没有值，所以把TMP[1]设置为SAL[1]。在设置了TMP[1]之后，下一行就是TMP[2]。计算第一个SAL[1]（由于ANY的当前值是2，因此SAL[CV()-1]的值是SAL[1]）。SAL[1]不为空，而且等于-1300，因此把TMP[2]设置为TMP[1]和SAL[2]的乘积。所有行都进行上述操作。

## MySQL、PostgreSQL 和 SQL Server

有关MySQL、PostgreSQL和SQL Server解决方案所采用的子查询方法的说明，请参阅本章第7.6节。

要注意，基于子查询解决方案的输出与 Oracle 和 DB2 解决方案的输出有少许差别，其原因来自 EMPNO 比较（它们按不同的顺序计算累乘积）。与累计和一样，其总数也是由标量子查询的谓词驱动的；在该解决方案中，行是按 EMPNO 排序的，而对于 Oracle/DB2 解决方案，行是按 SAL 排序的。

## 7.8 计算累计差

### 问题

对于数字列中的值，计算其累计差。例如，计算 DEPTNO 10 中工资的累计差。要返回下列结果集：

ENAME	SAL	RUNNING_DIFF
MILLER	1300	1300
CLARK	2450	-1150
KING	5000	-6150

### 解决方案

#### DB2 和 Oracle

使用窗口函数 SUM OVER 创建累计差：

```
1 select ename,sal,
2        sum(case when rn = 1 then sal else -sal end)
3          over(order by sal,empno) as running_diff
4   from (
5 select empno,ename,sal,
6        row_number()over(order by sal,empno) as rn
7   from emp
8  where deptno = 10
9        ) x
```

#### MySQL、PostgreSQL 和 SQL Server

使用标量子查询计算累计差：

```
1 select a.empno, a.ename, a.sal,
2        (select case when a.empno = min(b.empno) then sum(b.sal)
3                  else sum(-b.sal)
4                  end
5         from emp b
6        where b.empno <= a.empno
7              and b.deptno = a.deptno ) as rnk
8   from emp a
9  where a.deptno = 10
```

### 讨论

该解决方案与“生成累计和”一节介绍的解决方案大致相同。唯一的差别是：SAL 除了第一个值（因为要从 DEPTNO 10 的 SAL 开始）之外，其余所有值都返回负值。

## 7.9 计算模式

### 问题

查找某个列中值的模式（数学中的模式概念就是对于给定的数据集出现最频繁的元素）。例如，查找 DEPTNO 20 中工资的模式。例如下列工资：

```
select sal
  from emp
 where deptno = 20
 order by sal

      SAL
-----
      800
     1100
     2975
     3000
     3000
the mode is 3000.
```

### 解决方案

#### DB2 和 SQL Server

使用窗口函数 DENSE\_RANK，把工资重复出现次数分等级，以便提取模式：

```
1 select sal
2   from (
3 select sal,
4        dense_rank()over(order by cnt desc) as rnk
5   from (
6 select sal, count(*) as cnt
7   from emp
8  where deptno = 20
9  group by sal
10         ) x
11        ) y
12       ) y
13  where rnk = 1
```

#### Oracle

在 Oracle8i Database 中，可以使用 DB2 给出的解决方案。对于 Oracle9i 及更高版本，可以用聚集函数 MAX 的 KEEP 扩展，以得到 SAL 模式。特别要注意的是，如果存在绑带，也即多个行都是模式，则采用 KEEP 方案仅能得到一个，即其中工资最高的那个。如果想要看所有模式（如果存在多个模式），则必须修改该方案，或者简单地使用前面介绍的 DB2 解决方案。在这个例子中，由于 3000 是 DEPTNO 20 中 SAL 的模式，而且它也是最高的 SAL，因此以下方案就可以了：

```
1 select max(sal)
2        keep(dense_rank first order by cnt desc) sal
3   from (
4 select sal, count(*) cnt
5   from emp
6  where deptno=20
7  group by sal
8         )
```

## MySQL 和 PostgreSQL

使用子查询查找模式：

```

1 select sal
2   from emp
3  where deptno = 20
4  group by sal
5  having count(*) >= all ( select count(*)
6                           from emp
7                           where deptno = 20
8                           group by sal )

```

## 讨论

### DB2 和 SQL Server

内联视图 X 将返回每个 SAL 及它出现的次数。内联视图 Y 使用窗口函数 DENSE\_RANK (它允许绑带) 给结果排序。结果按每个 SAL 出现的次数分等级，如下所示：

```

1 select sal,
2        dense_rank()over(order by cnt desc) as rnk
3   from (
4 select sal,count(*) as cnt
5   from emp
6  where deptno = 20
7  group by sal
8 ) x

```

SAL	RNK
3000	1
800	2
1100	2
2975	2

最外层的查询只简单地保留 RNK 为 1 的行。

## Oracle

内联视图将返回所有 SAL 及其出现的次数，如下所示：

```

select sal, count(*) cnt
  from emp
 where deptno=20
 group by sal

```

SAL	CNT
800	1
1100	1
2975	1
3000	2

下一步，使用聚集函数 MAX 的 KEEP 扩展查找模式。如果仔细分析下面给出的 KEEP 子句，会发现它又有三个子句，即 DENSE\_RANK、FIRST 和 ORDER BY CNT DESC；

```
keep(dense_rank first order by cnt desc)
```

这种做法对求模式极其方便。KEEP 子句根据内联视图返回的 CNT 值来确定 MAX 返回

SAL 的哪个值。按从右向左的方向将 CNT 递减排序，然后保留下按 DENSE\_RANK 次序返回的所有 CNT 值的第一个值。查看一下内联视图的结果集，就会看到 3000 具有最高的 CNT 值——2。MAX(SAL) 返回的是拥有最高 CNT 值的最大 SAL，在本例中是 3000。

有关 Oracle 中集合函数的 KEEP 扩展的深入讨论，请参阅第 11 章第 11.11 节。

## MySQL 和 PostgreSQL

子查询将返回每个 SAL 出现的次数。外层查询将返回其的出现次数大于等于子查询所返回所有计数值的 SAL（换句话说，外层查询会返回 DEPTNO 20 中出现最多的工资）。

## 7.10 计算中间值

### 问题

计算一系列数字值的中间值（中间值就是一组有序元素中间成员的值）。例如，查找 DEPTNO 20 中工资的中间数。如下列工资：

```
select sal
  from emp
 where deptno = 20
 order by sal

      SAL
-----
      800
     1100
     2975
     3000
     3000
```

中间数为 2975。

### 解决方案

除了 Oracle 解决方案（用函数计算中间数）之外，其他所有解决方案都是以 Rozenshtein、Abramovich 和 Birger 在 *Optimizing Transact-SQL: Advanced Programming Techniques* (SQL Forum Press, 1997) 中描述的方法为基础的。与传统的自联接相比，窗口函数的引入，使解决方案更为有效。

## DB2

使用窗口函数 COUNT(\*) OVER 和 ROW\_NUMBER，查找中间数：

```
1 select avg(sal)
2   from (
3 select sal,
4        count(*) over() total,
5        cast(count(*) over() as decimal)/2 mid,
6        ceil(cast(count(*) over() as decimal)/2) next,
7        row_number() over (order by sal) rn
8   from emp
9  where deptno = 20
```



```

10      ) x
11  where ( mod(total,2) = 0
12         and rn in ( mid, mid+1 )
13      )
14      or ( mod(total,2) = 1
15         and rn = next
16      )

```

## MySQL 和 PostgreSQL

使用自联接查找中间数：

```

1  select avg(sal)
2  from (
3  select e.sal
4  from emp e, emp d
5  where e.deptno = d.deptno
6  and e.deptno = 20
7  group by e.sal
8  having sum(case when e.sal = d.sal then 1 else 0 end)
9         >= abs(sum(sign(e.sal - d.sal)))
10      )

```

## Oracle

使用函数 MEDIAN (Oracle Database 10g) 或 PERCENTILE\_CONT (Oracle9i Database);

```

1 select median (sal)
2   from emp
3  where deptno=20

1 select percentile_cont(0.5)
2       within group(order by sal)
3   from emp
4  where deptno=20

```

对于 Oracle8i Database, 使用 DB2 解决方案。对于 Oracle8i Database 之前的版本, 可以采用 PostgreSQL/MySQL 解决方案。

## SQL Server

使用窗口函数 COUNT(\*) OVER 和 ROW\_NUMBER, 可得到中间数:

```

1  select avg(sal)
2  from (
3  select sal,
4         count(*)over() total,
5         cast(count(*)over() as decimal)/2 mid,
6         ceiling(cast(count(*)over() as decimal)/2) next,
7         row_number()over(order by sal) rn
8  from emp
9  where deptno = 20
10     ) x
11  where ( total%2 = 0
12         and rn in ( mid, mid+1 )
13     )
14     or ( total%2 = 1
15         and rn = next
16     )

```

## 讨论

### DB2 和 SQL Server

DB2 和 SQL Server 解决方案的唯一差别是语法的稍许不同：SQL Server 用 “%” 求模，而 DB2 使用 MOD 函数；其余的都相同。内联视图 X 返回三个不同的计数值，TOTAL、MID 和 NEXT，还用到由 ROW\_NUMBER 生成的 RN。这些附加列有助于求解中间数。检验内联视图 X 的结果集，就会看到这些列表示的意义：

```
select sal,
       count(*)over() total,
       cast(count(*)over() as decimal)/2 mid,
       ceil(cast(count(*)over() as decimal)/2) next,
       row_number()over(order by sal) rn
from emp
where deptno = 20
```

SAL	TOTAL	MID	NEXT	RN
800	5	2.5	3	1
1100	5	2.5	3	2
2975	5	2.5	3	3
3000	5	2.5	3	4
3000	5	2.5	3	5

要得到中间数，一定要把 SAL 值由低到高排序。由于 DEPTNO 20 中的职员数是奇数，因此它的中间数就是其 RN 与 NEXT 相等的 SAL（即大于职员总数除以 2 的最小整数）。

如果结果集返回奇数行，WHERE 子句的第一部分（第 11~13 行）条件不满足。如果能够确定结果集是奇数行，则可以简化为：

```
select avg(sal)
from (
select sal,
       count(*)over() total,
       ceil(cast(count(*)over() as decimal)/2) next,
       row_number()over(order by sal) rn
from emp
where deptno = 20
) x
where rn = next
```

令人遗憾的是，如果结果集包含偶数行，上述简化的解决方案就行不通。在最初的解决方案中，采用 MID 列中的值处理偶数行。想想 DEPTNO 30 的内联视图 X 的结果会怎样，该部门有 6 名职员：

```
select sal,
       count(*)over() total,
       cast(count(*)over() as decimal)/2 mid,
       ceil(cast(count(*)over() as decimal)/2) next,
       row_number()over(order by sal) rn
from emp
where deptno = 30
```

SAL	TOTAL	MID	NEXT	RN
950	6	3	3	1
1250	6	3	3	2
1250	6	3	3	3

1500	6	3	3	4
1600	6	3	3	5
2850	6	3	3	6

由于返回了偶数行，则采用下述方式计算中间数：计算 RN 分别等于 MID 和 MID + 1 两行的平均数。

## MySQL 和 PostgreSQL

根据第一个自联接表 EMP 计算中间数，而该表返回了所有工资的笛卡儿积 (GROUP BY E.SAL 会去掉重复值)。HAVING 子句使用函数 SUM 计算 E.SAL 等于 D.SAL 的次数；如果这个值大于等于 E.SAL 且大于 D.SAL 次数，那么该行就是中间数。在 SELECT 列表中加入 SUM 就可以观察到这种情况：

```
select e.sal,
       sum(case when e.sal=d.sal
                 then 1 else 0 end) as cnt1,
       abs(sum(sign(e.sal - d.sal))) as cnt2
  from emp e, emp d
 where e.deptno = d.deptno
    and e.deptno = 20
 group by e.sal
```

SAL	CNT1	CNT2
800	1	4
1100	1	2
2975	1	0
3000	4	6

## Oracle

在 Oracle Database 10g 或 Oracle9i Database 中，可以使用 Oracle 提供的函数计算中间数；对于 Oracle8i Database，可以采用 DB2 解决方案；其他版本必须采用 PostgreSQL 解决方案。显然可以用 MEDIAN 函数计算中间值，用 PERCENTILE\_CONT 函数也可以计算中间值就不那么显而易见了。传递给 PERCENTILE\_CONT 的值 0.5 是一个百分比值。子句 WITHIN GROUP (ORDER BY SAL) 确定 PERCENTILE\_CONT 要搜索哪些有序行（记住，中间值就是一组已排序值的中间值）。返回的值就是搜索的有序行中符合给定百分比（在这个例子中是 0.5，因为其两个边界值分别为 0 和 1）的值。

## 7.11 求总和的百分比

### 问题

求特定列中的值占总和的百分比。例如，确定所有 DEPTNO 10 工资占总工资的百分比 (DEPTNO 10 的工资在总工资中的百分比数)。

### 解决方案

总的来说，在 SQL 中计算占总数的百分比跟书面计算一样：先除后乘。这个例子要计算

表 EMP 中 DEPTNO 10 工资所占的百分比。首先，算出 DEPTNO 10 的工资，然后除以表中的工资总和，最后一步，乘以 100，则返回一个表示百分比的值。

## MySQL 和 PostgreSQL

DEPTNO 10 的工资总和除以所有工资总和：

```
1 select (sum(
2     case when deptno = 10 then sal end)/sum(sal)
3     )*100 as pct
4   from emp
```

## DB2、Oracle 和 SQL Server

使用内联视图及窗口函数 SUM OVER，计算出所有工资总和以及 DEPTNO 10 的工资和。然后，在外层查询中进行除法和乘法操作：

```
1 select distinct (d10/total)*100 as pct
2   from (
3     select deptno,
4            sum(sal)over() total,
5            sum(sal)over(partition by deptno) d10
6     from emp
7     ) x
8  where deptno=10
```

## 讨论

### MySQL 和 PostgreSQL

用 CASE 语句能够轻松地得到 DEPTNO 10 的工资。然后将它们加起来，并除以所有工资总和。由于聚集时会忽略 NULL 值，所以 CASE 语句中不必加入 ELSE 子句。如果想看到确切的被除数和除数，则可以执行不做除法的查询：

```
select sum(case when deptno = 10 then sal end) as d10,
       sum(sal)
  from emp

D10  SUM(SAL)
----  -
8750          29025
```

依定义 SAL 的方式不同，在进行除法操作时可能需要做显式类型转换。例如，在 DB2、SQL Server 和 PostgreSQL 中，如果 SAL 定义为整数，则可以把它转换为小数，以便得到正确答案，如下所示：

```
select (cast(
       sum(case when deptno = 10 then sal end)
         as decimal)/sum(sal)
       )*100 as pct
  from emp
```

### DB2、Oracle 和 SQL Server

除传统解决方案外，该方案使用窗口函数计算相对于总数的百分数。对于 DB2 和 SQL Server，如果 SAL 定义为整数类型，则在除法操作之前，需要进行类型转换：

```

select distinct
  cast(d10 as decimal)/total*100 as pct
  from (
select deptno,
  sum(sal)over() total,
  sum(sal)over(partition by deptno) d10
  from emp
  ) x
where deptno=10

```

必须记住，窗口函数在 WHERE 子句后执行。因此不能把针对 DEPTNO 的筛选放在内联视图 X 中。分别考虑一下内联视图 X 中包含及不包含 DEPTNO 筛选的结果。首先，看一下不包含 DEPTNO 筛选的结果：

```

select deptno,
  sum(sal)over() total,
  sum(sal)over(partition by deptno) d10
  from emp

```

DEPTNO	TOTAL	D10
10	29025	8750
10	29025	8750
10	29025	8750
20	29025	10875
20	29025	10875
20	29025	10875
20	29025	10875
20	29025	10875
30	29025	9400
30	29025	9400
30	29025	9400
30	29025	9400
30	29025	9400
30	29025	9400

包含 DEPTNO 筛选的结果：

```

select deptno,
  sum(sal)over() total,
  sum(sal)over(partition by deptno) d10
  from emp
 where deptno=10

```

DEPTNO	TOTAL	D10
10	8750	8750
10	8750	8750
10	8750	8750

由于窗口函数在 WHERE 子句后执行，因此 TOTAL 的值仅表示 DEPTNO 10 的工资之和，而实际上需要用 TOTAL 表示所有工资的总和。这就是必须把针对 DEPTNO 的筛选放在内联视图 X 外面的原因。

## 7.12 对可空列作聚集

### 问题

对某列进行聚集运算，但该列的值可为空，由于函数会忽略 NULL 值，能否保持聚集运算的准确性令人担忧。例如，想要求 DEPTNO 30 中职员平均佣金，但有些职员不挣

佣金（这些职员的 COMM 值为 NULL）。由于聚集运算会忽略 NULL，因此输出结果的准确性没有保障。在进行聚集运算时有时可能需要以某种方式将 NULL 值包括进来。

## 解决方案

使用 COALESCE 函数把 NULL 转换为 0，这样在进行聚集时可以把它们包括进来：

```
1 select avg(coalesce(comm,0)) as avg_comm
2   from emp
3  where deptno=30
```

## 讨论

请务必记住，在使用聚集函数时会忽略 NULL。不使用 COALESCE 函数时该解决方案的输出如下：

```
select avg(comm)
  from emp
 where deptno=30

AVG(COMM)
-----
      550
```

该查询表明，DEPTNO 30 的平均佣金是 550，快速检查这些行如下：

```
select ename, comm
  from emp
 where deptno=30
 order by comm desc

ENAME          COMM
-----
BLAKE
JAMES
MARTIN          1400
WARD            500
ALLEN           300
TURNER           0
```

这表明六个职员中只有四个职员挣得佣金。DEPTNO 30 中所有佣金的总和是 2200，其平均值应该是 2200/6，而不是 2200/4。如果不用 COALESCE 函数，回答的是问题“DEPTNO 30 中挣得佣金的职员的平均佣金是多少？”，而不是“DEPTNO 30 中所有职员的平均佣金是多少？”。使用聚集时记住要相应处理 NULL 值。

## 7.13 计算不包含最大值和最小值的均值

### 问题

计算平均数，但希望排除最大和最小值，以（希望能）减少数据畸偏造成的影响。例如，计算除最高和最低工资外的所有职员的平均工资。

## 解决方案

### MySQL 和 PostgreSQL

使用子查询排除最高和最低值：

```

1  select avg(sal)
2    from emp
3   where sal not in (
4     (select min(sal) from emp),
5     (select max(sal) from emp)
6   )

```

### DB2、Oracle 和 SQL Server

使用内联视图及窗口函数 MAX OVER 和 MIN OVER，生成一个结果集，可以很容易地从中剔除最大和最小值：

```

1  select avg(sal)
2    from (
3   select sal, min(sal)over() min_sal, max(sal)over() max_sal
4     from emp
5    ) x
6   where sal not in (min_sal,max_sal)

```

## 讨论

### MySQL 和 PostgreSQL

子查询返回表中的最高工资和最低工资。针对返回的值使用 NOT IN，就可以从平均值中排除最高工资和最低工资。记住，如果存在重复（多个职员都是最高或最低工资），那么他们都会被排除在平均值之外。如果只想排除一个最高和最低值，只需从 SUM 中减去它们，再做除法：

```

select (sum(sal)-min(sal)-max(sal))/(count(*)-2)
  from emp

```

### DB2、Oracle 和 SQL Server

内联视图 X 将返回所有工资，其中包括最高工资和最低工资：

```

select sal, min(sal)over() min_sal, max(sal)over() max_sal
  from emp

```

SAL	MIN_SAL	MAX_SAL
800	800	5000
1600	800	5000
1250	800	5000
2975	800	5000
1250	800	5000
2850	800	5000
2450	800	5000
3000	800	5000
5000	800	5000
1500	800	5000
1100	800	5000
950	800	5000
3000	800	5000

1300            800            5000

从每一行都可以访问最高工资和最低工资，因此，要找出哪些工资是最高工资的和/或最低工资的非常简单。外层查询会对内联视图X返回的行作筛选，这样，所有与MIN\_SAL和MAX\_SALAN相匹配的行都会从平均值中排除掉。

## 7.14 把字母数字串转换为数值

### 问题

对于字母数字的数据，只返回数字值。从字符串“paul123f321”中返回123321。

### 解决方案

#### DB2

使用函数TRANSLATE和REPLACE，从字母数字串中提取数字字符：

```
1 select cast(
2     replace(
3         translate( 'paul123f321',
4                     repeat('#',26),
5                     'abcdefghijklmnopqrstuvwxyz'), '#', '')
6     as integer ) as num
7 from t1
```

#### Oracle 和 PostgreSQL

使用函数TRANSLATE和REPLACE，可以从包含字母数字的字符串中提取数字字符：

```
1 select cast(
2     replace(
3         translate( 'paul123f321',
4                     'abcdefghijklmnopqrstuvwxyz',
5                     rpad('#',26,'#')), '#', '')
6     as integer ) as num
7 from t1
```

#### MySQL 和 SQL Server

到本书编写时为止，这两个供应商都不支持TRANSLATE函数，因此这里不能给出解决方案了。

### 讨论

两种解决方案的唯一差别是语法，DB2使用函数REPEAT代替RPAD，而且TRANSLATE参数列表的顺序也不同。以下的解释采用了Oracle/PostgreSQL解决方案，DB2也类似。如果从里向外运行该查询（仅仅从TRANSLATE开始），就会发现这非常简单。首先，TRANSLATE把非数字字符转换为“#”：

```
select translate( 'paul123f321',
                  'abcdefghijklmnopqrstuvwxyz',
                  rpad('#',26,'#')) as num
```



```

from t1
NUM
-----
####123#321

```

由于现在所有非数字字符都用“#”表示了，因此只需使用REPLACE去掉它们，然后把结果转换为数值。这个特殊的例子尤其简单，因为字符串中只有字母和数字。如果还有其他字符，那么用另一种方法会更容易：不是找出非数字字符并去掉它们，而是找出所有数字字符，并去掉不属于这些字符范围的其他字符。下面的例子会有助于理解这种技巧：

```

select replace(
  translate('paul123f321',
    replace(translate( 'paul123f321',
      '0123456789',
      rpad('#',10,'#')), '#',''),
    rpad('#',length('paul123f321'),'#'),'#','') as num
from t1
NUM
-----
123321

```

较之原始方案，该解决方案看起来有点儿费解，但如果把它分解开来就容易理解了。观察一下最内层的TRANSLATE调用：

```

select translate( 'paul123f321',
  '0123456789',
  rpad('#',10,'#'))
from t1
TRANSLATE('
-----
paul###f###

```

与原来方案不同的是，它没有用“#”字符替换每个非数字字符，而是用“#”字符替换所有数字字符。接下来，去掉所有“#”，这样，只剩下非数字字符：

```

select replace(translate( 'paul123f321',
  '0123456789',
  rpad('#',10,'#')), '#','')
from t1
REPLA
-----
paulf

```

下一步，再次调用TRANSLATE，这次用“#”字符替换原始字符串中的所有非数字字符（前面查询的结果）：

```

select translate('paul123f321',
  replace(translate( 'paul123f321',
    '0123456789',
    rpad('#',10,'#')), '#',''),
  rpad('#',length('paul123f321'),'#'))
from t1
TRANSLATE('
-----
####123#321

```

到这里停一停，检验一下最外层的 TRANSLATE 调用。RPAD 的第二个参数（DB2 中 REPEAT 的第二个参数）是原始字符串的长度。这样做很方便，因为没有任何字符出现的次数会比它所在的整个字符串长。现在，用“#”字符替换所有非数字字符；最后一步，使用 REPLACE 去掉所有“#”。至此，只剩下数字。

## 7.15 更改累计和中的值

### 问题

根据另一列中的值修改累计和中的值。假设一个场景，要显示信用卡账号的事务处理历史以及每次事务处理之后的当前余额。在这个例子中，将使用下面给出的视图 V：

```
create view V (id,amt,trx)
as
select 1, 100, 'PR' from t1 union all
select 2, 100, 'PR' from t1 union all
select 3, 50, 'PY' from t1 union all
select 4, 100, 'PR' from t1 union all
select 5, 200, 'PY' from t1 union all
select 6, 50, 'PY' from t1

select * from V
```

ID	AMT	TR
1	100	PR
2	100	PR
3	50	PY
4	100	PR
5	200	PY
6	50	PY

ID 列唯一标识每次事务处理。AMT 列表示每次事务处理（取款或存款）涉及的金额。TRX 列定义了事务处理的类型；取款是“PY”，存款是“PR”。如果 TRX 值是 PY，则想要从累计和中减去 AMT 值代表的金额；如果 TRX 值是 PR，则想要给累计和加上 AMT 值代表的金额。最后应该返回如下结果集：

TRX_TYPE	AMT	BALANCE
PURCHASE	100	100
PURCHASE	100	200
PAYMENT	50	150
PURCHASE	100	250
PAYMENT	200	50
PAYMENT	50	0

### 解决方案

#### DB2 和 Oracle

使用窗口函数 SUM OVER 创建累计和，并使用 CASE 表达式判断事务处理的类型：

```
1 select case when trx = 'PY'
2             then 'PAYMENT'
3             else 'PURCHASE'
4             end trx_type,
```

```
5      amt,  
6      sum(  
7          case when trx = 'PY'  
8              then -amt else amt  
9          end  
10     ) over (order by id,amt) as balance  
11 from V
```

## MySQL、PostgreSQL 和 SQL Server

使用标量子查询创建累计和，并使用 CASE 表达式判断事务处理的类型：

```
1 select case when v1.trx = 'PY'  
2     then 'PAYMENT'  
3     else 'PURCHASE'  
4 end as trx_type,  
5 v1.amt,  
6 (select sum(  
7     case when v2.trx = 'PY'  
8         then -v2.amt else v2.amt  
9     end  
10 )  
11 from V v2  
12 where v2.id <= v1.id) as balance  
13 from V v1
```

## 讨论

CASE 表达式判断是该给累计和加上当前的 AMT 值还是从中减去当前的 AMT 值。如果事务处理是取款，则把 AMT 更改为负值，这样就减少了累计和。CASE 表达式的结果如下所示：

```
select case when trx = 'PY'  
    then 'PAYMENT'  
    else 'PURCHASE'  
end as trx_type,  
case when trx = 'PY'  
    then -amt else amt  
end as amt  
from V
```

TRX_TYPE	AMT
PURCHASE	100
PURCHASE	100
PAYMENT	-50
PURCHASE	100
PAYMENT	-200
PAYMENT	-50

在确定了事务处理类型之后，就可以从累计和中加上或者减去 AMT 值。有关窗口函数 SUM OVER 或标量子查询如何创建累计和的说明，请参阅“计算累计和”一节。

## 第 8 章

# 日期运算

本章将介绍进行简单日期运算的技巧，包括常见的操作，如给日期增加天数、计算两个日期之间的工作日数以及计算两个日期之间相差的天数等。

如果能够利用 RDBMS 的内置函数成功地处理日期，可以大大提高效率。本章中的问题极力充分利用各 RDBMS 的内置函数。另外，对所有问题，都选用同一种日期格式，即“DD-MON-YYYY”。这样做，对使用过一个 RDBMS 并想学习其他 RDBMS 的人会有益处；采用同一种格式，将会有助于读者将精力集中在每个 RDBMS 提供的不同技巧和函数上，而不必考虑默认的日期格式是什么。

**注意：**本章重点介绍基本日期运算，下一章将讲述更多的高级日期处理。本章列出的问题都使用了简单日期类型，如果使用更复杂的日期类型，就应该相应地调整解决方案。

## 8.1 加减日、月、年

### 问题

对日期加减日、月、年。例如，根据员工 CLARK 的 HIREDATE（聘用日期），计算另外 6 个不同的日期：聘用 CLARK 之前及之后的 5 天；聘用 CLARK 之前及之后的 5 个月；聘用 CLARK 之前及之后的 5 年。例如，聘用 CLARK 的日期为“09-JUN-1981”，要求返回如下结果集：

HD_MINUS_5D	HD_PLUS_5D	HD_MINUS_5M	HD_PLUS_5M	HD_MINUS_5Y	HD_PLUS_5Y
04-JUN-1981	14-JUN-1981	09-JAN-1981	09-NOV-1981	09-JUN-1976	09-JUN-1986
12-NOV-1981	22-NOV-1981	17-JUN-1981	17-APR-1982	17-NOV-1976	17-NOV-1986
18-JAN-1982	28-JAN-1982	23-AUG-1981	23-JUN-1982	23-JAN-1977	23-JAN-1987

## 解决方案

### DB2

对日期值，允许进行标准的加、减操作，但是，如果对日期进行加减操作，后面一定要给出它所表示的时间单位：

```
1 select hiredate -5 day as hd_minus_5D,
2        hiredate +5 day as hd_plus_5D,
3        hiredate -5 month      as hd_minus_5M,
4        hiredate +5 month      as hd_plus_5M,
5        hiredate -5 yearas hd_minus_5Y,
6        hiredate +5 yearas hd_plus_5Y
7   from emp
8  where deptno = 10
```

### Oracle

对天数采用标准加减，而使用 ADD\_MONTHS 函数加减月数和年数：

```
1 select hiredate-5as hd_minus_5D,
2        hiredate+5as hd_plus_5D,
3        add_months(hiredate,-5)      as hd_minus_5M,
4        add_months(hiredate,5) as hd_plus_5M,
5        add_months(hiredate,-5*12)   as hd_minus_5Y,
6        add_months(hiredate,5*12)    as hd_plus_5Y
7   from emp
8  where deptno = 10
```

### PostgreSQL

同时使用标准加减与INTERVAL关键字，INTERVAL指定时间单位。在指定INTERVAL值时，需要用单引号：

```
1 select hiredate - interval '5 day' as hd_minus_5D,
2        hiredate + interval '5 day' as hd_plus_5D,
3        hiredate - interval '5 month'      as hd_minus_5M,
4        hiredate + interval '5 month'      as hd_plus_5M,
5        hiredate - interval '5 year'as hd_minus_5Y,
6        hiredate + interval '5 year'as hd_plus_5Y
7   from emp
8  where deptno=10
```

### MySQL

同时使用标准加减与INTERVAL关键字，INTERVAL指定时间单位。它不同于PostgreSQL 解决方案，不必在INTERVAL值周围加单引号：

```
1 select hiredate - interval 5 day      as hd_minus_5D,
2        hiredate + interval 5 day      as hd_plus_5D,
3        hiredate - interval 5 month    as hd_minus_5M,
4        hiredate + interval 5 month    as hd_plus_5M,
5        hiredate - interval 5 year     as hd_minus_5Y,
6        hiredate + interval 5 year     as hd_plus_5Y
7   from emp
8  where deptno=10
```

另外，也可以使用 DATE\_ADD 函数，如下所示：

```
1 select date_add(hiredate,interval -5 day) as hd_minus_5D,
```

```

2      date_add(hiredate,interval 5 day) as hd_plus_5D,
3      date_add(hiredate,interval -5 month) as hd_minus_5M,
4      date_add(hiredate,interval 5 month) as hd_plus_5M,
5      date_add(hiredate,interval -5 year) as hd_minus_5Y,
6      date_add(hiredate,interval 5 year) as hd_plus_5DY
7  from emp
8  where deptno=10

```

## SQL Server

使用 DATEADD 函数，可以对日期进行不同时间单位的加减操作：

```

1 select dateadd(day,-5,hiredate) as hd_minus_5D,
2        dateadd(day,5,hiredate) as hd_plus_5D,
3        dateadd(month,-5,hiredate) as hd_minus_5M,
4        dateadd(month,5,hiredate) as hd_plus_5M,
5        dateadd(year,-5,hiredate) as hd_minus_5Y,
6        dateadd(year,5,hiredate) as hd_plus_5Y
7  from emp
8  where deptno = 10

```

## 讨论

当进行日期运算时，Oracle 解决方案采用整型值表示天数。然而，这种方法只能用来对 DATE 类型进行运算。Oracle9i Database 引入了 TIMESTAMP 类型，对这种类型的值应该使用 PostgreSQL 给出的 INTERVAL 解决方案；还要注意，不要把 TIMESTAMP 传递给老版本的日期函数，如 ADD\_MONTHS，因为这样做会丢失 TIMESTAMP 值可能包含的小数部分。

INTERVAL 关键字以及与它一起使用的字符串符合 ISO 标准的 SQL 语法，该标准要求间隔时间单位用单引号括起来。PostgreSQL（以及 Oracle9i Database，或更高版本）遵从此标准。但由于 MySQL 不支持引号，因此在某种程度上它没有遵从此标准。

## 8.2 计算两个日期之间的天数

### 问题

求两个日期之间相差的天数。例如，想了解员工 ALLEN 和员工 WARD 的 HIREDATE（聘用日期）之间相差的天数。

### 解决方案

#### DB2

使用两个内联视图求 WARD 和 ALLEN 的 HIREDATE（聘用日期）。然后使用 DAYS 函数从一个 HIREDATE 中减去另一个 HIREDATE：

```

1 select days(ward_hd) - days(allen_hd)
2   from (
3 select hiredate as ward_hd
4   from emp
5  where ename = 'WARD'
6   ) x,

```

```
7      (
8  select hiredate as allen_hd
9      from emp
10     where ename = 'ALLEN'
11      ) y
```

## Oracle 和 PostgreSQL

使用两个内联视图求 WARD 和 ALLEN 的 HIREDATE（聘用日期）。然后从一个日期中减去另一个日期：

```
1  select ward_hd - allen_hd
2      from (
3  select hiredate as ward_hd
4      from emp
5     where ename = 'WARD'
6      ) x,
7      (
8  select hiredate as allen_hd
9      from emp
10     where ename = 'ALLEN'
11      ) y
```

## MySQL 和 SQL Server

使用 DATEDIFF 函数可以求两个日期之间相差的天数。MySQL 中的 DATEDIFF 函数仅需要两个参数（即要计算相差天数的两个日期），第一个参数应是两个日期中较小的值，以避免出现负值（SQL Server 中正好相反）。在 SQL Server 中，可以指定该函数返回值所表示的类型（在这个例子中，返回以“日”为单位的差）。下面的解决方案采用了 SQL Server 的版本：

```
1  select datediff(day,allen_hd,ward_hd)
2      from (
3  select hiredate as ward_hd
4      from emp
5     where ename = 'WARD'
6      ) x,
7      (
8  select hiredate as allen_hd
9      from emp
10     where ename = 'ALLEN'
11      ) y
```

MySQL 用户只需去掉该函数的第一个参数，交换一下传递 ALLEN\_HD 和 WARD\_HD 的顺序即可。

## 讨论

对于所有的解决方案，内联视图 X 和 Y 分别返回员工 WARD 和 ALLEN 的 HIREDATE。例如：

```
select ward_hd, allen_hd
      from (
select hiredate as ward_hd
      from emp
     where ename = 'WARD'
      ) y,
```

```
(
select hiredate as allen_hd
  from emp
 where ename = 'ALLEN'
) x
```

```
WARD_HD      ALLEN_HD
-----
22-FEB-1981  20-FEB-1981
```

注意：这里产生了笛卡儿积，因为没有在 X 和 Y 之间指定联接条件。在这种特定情况下，缺少连接并没有害处，因为 X 和 Y 的基数都为 1，这样结果集最终就只有一行（很显然， $1 \times 1 = 1$ ）。要得到相差的天数，只需使用相应数据库中方法，将返回的两个值相减即可。

## 8.3 确定两个日期之间的工作日数目

### 问题

给定两个日期，求它们之间（包括这两个日期本身）有多少个“工作”日。例如，如果 1 月 10 日是星期一，1 月 11 日是星期二，由于这两个日期是典型的工作日，所以两个日期之间的工作日数是 2。对于这个问题，“工作日”定义为非周六/周日的日子。

### 解决方案

下面的例子计算 BLAKE 和 JONES 的 HIREDATE（聘用日期）之间的工作日数。要确定两个日期之间的工作日数，可以使用基干表，对两个日期（其中包括起始日期和结束日期）之间的每一天都返回一行。此后，计算工作日数只是数一下返回的日期中非周六/周日的数目。

---

注意：如果也想去掉假期，则可以创建一个 HOLIDAYS 表。然后引入 NOT IN 前缀，就能够从解决方案中去掉 HOLIDAYS 中列出的日子。

---

### DB2

使用基干表 T500，能够生成两个日期之间包含的行数（表示天数），然后，对非周末的所有日期计数。使用 DAYNAME 函数可返回每个日期是星期几。例如：

```
1 select sum(case when dayname(jones_hd+t500.id day -1 day)
2               in ( 'Saturday','Sunday' )
3                   then 0 else 1
4               end) as days
5   from (
6 select max(case when ename = 'BLAKE'
7               then hiredate
8               end) as blake_hd,
9        max(case when ename = 'JONES'
10              then hiredate
11              end) as jones_hd
12   from emp
13  where ename in ( 'BLAKE','JONES' )
14         ) x,
15        t500
```



```
16 where t500.id <= blake_hd-jones_hd+1
```

## MySQL

使用基干表 T500 能够生成两个日期之间包含的行数（天数），然后，对非周末的所有日期计数。使用 DATE\_ADD 函数可以给每个日期增加天数，使用 DATE\_FORMAT 函数判定每个日期是星期几：

```
1 select sum(case when date_format(
2                     date_add(jones_hd,
3                               interval t500.id-1 DAY), '%a')
4                     in ( 'Sat', 'Sun' )
5                     then 0 else 1
6                     end) as days
7   from (
8     select max(case when ename = 'BLAKE'
9                   then hiredate
10                  end) as blake_hd,
11          max(case when ename = 'JONES'
12                  then hiredate
13                  end) as jones_hd
14     from emp
15    where ename in ( 'BLAKE', 'JONES' )
16          ) x,
17         t500
18  where t500.id <= datediff(blake_hd, jones_hd)+1
```

## Oracle

使用基干表 T500 能够生成两个日期之间包含的行数（天数），然后，对非周末的所有日期计数。使用 TO\_CHAR 函数确定每个日期是星期几：

```
1 select sum(case when to_char(jones_hd+t500.id-1, 'DY')
2                     in ( 'SAT', 'SUN' )
3                     then 0 else 1
4                     end) as days
5   from (
6     select max(case when ename = 'BLAKE'
7                   then hiredate
8                   end) as blake_hd,
9          max(case when ename = 'JONES'
10                 then hiredate
11                 end) as jones_hd
12     from emp
13    where ename in ( 'BLAKE', 'JONES' )
14          ) x,
15         t500
16  where t500.id <= blake_hd-jones_hd+1
```

## PostgreSQL

使用基干表 T500 能够生成两个日期之间包含的行数（天数），然后，对非周末的所有日期计数。使用 TO\_CHAR 函数确定每个日期是星期几：

```
1 select sum(case when trim(to_char(jones_hd+t500.id-1, 'DAY'))
2                     in ( 'SATURDAY', 'SUNDAY' )
3                     then 0 else 1
4                     end) as days
5   from (
6     select max(case when ename = 'BLAKE'
7                   then hiredate
```

```

8          end) as blake_hd,
9          max(case when ename = 'JONES'
10                then hiredate
11                end) as jones_hd
12    from emp
13   where ename in ( 'BLAKE','JONES' )
14          ) x,
15          t500
16   where t500.id <= blake_hd-jones_hd+1

```

## SQL Server

使用基于表 T500 生成两个日期之间包含的行数（天数），然后，对非周末的所有日期计数。使用 DATENAME 函数确定每个日期是星期几：

```

1  select sum(case when datename(dw,jones_hd+t500.id-1)
2                  in ( 'SATURDAY','SUNDAY' )
3                  then 0 else 1
4                  end) as days
5  from (
6  select max(case when ename = 'BLAKE'
7                then hiredate
8                end) as blake_hd,
9          max(case when ename = 'JONES'
10                then hiredate
11                end) as jones_hd
12  from emp
13  where ename in ( 'BLAKE','JONES' )
14          ) x,
15          t500
16  where t500.id <= datediff(day,jones_hd-blake_hd)+1

```

## 讨论

尽管每个 RDBMS 都需要使用不同的内置函数确定日期名，对每个系统来说，总体解决方案都相同。可以把解决方案分成下面两个步骤：

1. 返回起始日期和结束日期之间的天数（二者均包含在内）；
2. 计数除周末以外共有多少天（即行数）。

内联视图 X 完成第一步操作。如果检查内联视图 X，会注意到，它使用了聚集函数 MAX，该函数用于删除 NULL。如果对 MAX 的用法不是很清楚，下面这个例子会有助于读者理解。以下显示了未使用 MAX 时内联视图 X 的结果：

```

select case when ename = 'BLAKE'
            then hiredate
            end as blake_hd,
       case when ename = 'JONES'
            then hiredate
            end as jones_hd
  from emp
 where ename in ( 'BLAKE','JONES' )

```

BLAKE_HD	JONES_HD
01-MAY-1981	02-APR-1981

如果不使用 MAX, 会返回两行。而使用 MAX, 只返回一行, 且去掉了 NULL:

```
select max(case when ename = 'BLAKE'
               then hiredate
            end) as blake_hd,
       max(case when ename = 'JONES'
               then hiredate
            end) as jones_hd
  from emp
 where ename in ( 'BLAKE','JONES' )
```

```
BLAKE_HD      JONES_HD
-----
01-MAY-1981  02-APR-1981
```

上述两个日期之间的天数 (其中包括两个日期本身) 是 30。既然两个日期处于一行, 那么, 下一步就要对这 30 天的每一天分别生成一行记录。要返回 30 天 (行), 需使用表 T500。由于表 T500 中的每个 ID 值都比它的前一个值大 1, 在两个日期中较早的一个 (JONES\_HD) 上分别加上 T500 中各行的 ID, 就可以生成从 JONES\_HD 日期开始直到 BLAKE\_HD (包括) 的连续工作日。其结果如下所示 (使用 Oracle 语法):

```
select x.*, t500.*, jones_hd+t500.id-1
  from (
select max(case when ename = 'BLAKE'
               then hiredate
            end) as blake_hd,
       max(case when ename = 'JONES'
               then hiredate
            end) as jones_hd
  from emp
 where ename in ( 'BLAKE','JONES' )
    ) x,
    t500
 where t500.id <= blake_hd-jones_hd+1
```

BLAKE_HD	JONES_HD	ID	JONES_HD+T5
01-MAY-1981	02-APR-1981	1	02-APR-1981
01-MAY-1981	02-APR-1981	2	03-APR-1981
01-MAY-1981	02-APR-1981	3	04-APR-1981
01-MAY-1981	02-APR-1981	4	05-APR-1981
01-MAY-1981	02-APR-1981	5	06-APR-1981
01-MAY-1981	02-APR-1981	6	07-APR-1981
01-MAY-1981	02-APR-1981	7	08-APR-1981
01-MAY-1981	02-APR-1981	8	09-APR-1981
01-MAY-1981	02-APR-1981	9	10-APR-1981
01-MAY-1981	02-APR-1981	10	11-APR-1981
01-MAY-1981	02-APR-1981	11	12-APR-1981
01-MAY-1981	02-APR-1981	12	13-APR-1981
01-MAY-1981	02-APR-1981	13	14-APR-1981
01-MAY-1981	02-APR-1981	14	15-APR-1981
01-MAY-1981	02-APR-1981	15	16-APR-1981
01-MAY-1981	02-APR-1981	16	17-APR-1981
01-MAY-1981	02-APR-1981	17	18-APR-1981
01-MAY-1981	02-APR-1981	18	19-APR-1981
01-MAY-1981	02-APR-1981	19	20-APR-1981
01-MAY-1981	02-APR-1981	20	21-APR-1981
01-MAY-1981	02-APR-1981	21	22-APR-1981
01-MAY-1981	02-APR-1981	22	23-APR-1981
01-MAY-1981	02-APR-1981	23	24-APR-1981
01-MAY-1981	02-APR-1981	24	25-APR-1981
01-MAY-1981	02-APR-1981	25	26-APR-1981
01-MAY-1981	02-APR-1981	26	27-APR-1981

01-MAY-1981	02-APR-1981	27	28-APR-1981
01-MAY-1981	02-APR-1981	28	29-APR-1981
01-MAY-1981	02-APR-1981	29	30-APR-1981
01-MAY-1981	02-APR-1981	30	01-MAY-1981

检查 WHERE 子句会注意到，对 BLAKE\_HD 和 JONES\_HD 之差进行了加 1 操作，生成了 30 行（否则的话，将生成 29 行）。另外，还会看到，对外部查询的 SELECT 列表中的 T500.ID 进行了减 1 操作，这是由于 ID 的起始值为 1，而且，对 JONES\_HD 进行加 1 操作，会导致从最终结果中减掉 JONES\_HD。

一旦生成了结果集需要的行数，可以使用 CASE 表达式“标记”返回的每一天是工作日还是周末（工作日返回 1，周末返回 0）。最后一步，使用求和函数 SUM 计算 1 的个数，以得到最终答案。

## 8.4 确定两个日期之间的月份数或年数

### 问题

求两个日期之间相差的月数或年数。例如，求第一个员工和最后一个员工聘用之间相差的月份数，以及这些月折合的年数。

### 解决方案

由于一年有 12 个月，因此，获得两个日期之间的月份数之后，再除以 12，就能得到年数。在有了相应的解决方案后，可以根据此年数的不同用途对结果进行舍/入。例如，表 EMP 中的第一个 HIREDATE（聘用日期）是“17-DEC-1980”，最后一个 HIREDATE 是“12-JAN-1983”。如果对年进行减法运算（1983 减去 1980），结果是 3 年。然而，月份差大约为 25（两年多一点儿）。所以应该修改解决方案。下列的解决方案返回的结果是 25 个月及 2 年。

### DB2 和 MySQL

使用函数 YEAR 和 MONTH 为给定日期返回 4 位数的年份和两位数的月份：

```

1  select mnth, mnth/12
2      from (
3  select (year(max_hd) - year(min_hd))*12 +
4         (month(max_hd) - month(min_hd)) as mnth
5      from (
6  select min(hiredate) as min_hd, max(hiredate) as max_hd
7      from emp
8      ) x
9      ) y

```

### Oracle

使用函数 MONTHS\_BETWEEN，将得到两个日期之间相差的月数（要得到相差年数，只需除以 12 即可）：

```

1  select months_between(max_hd,min_hd),
2     months_between(max_hd,min_hd)/12
3  from (
4  select min(hiredate) min_hd, max(hiredate) max_hd
5  from emp
6  ) x

```

## PostgreSQL

使用函数 EXTRACT，为给定日期返回 4 位数的年和两位数的月：

```

1  select mnth, mnth/12
2  from (
3  select ( extract(year from max_hd) -
4         extract(year from min_hd) ) * 12
5         +
6         ( extract(month from max_hd) -
7         extract(month from min_hd) ) as mnth
8  from (
9  select min(hiredate) as min_hd, max(hiredate) as max_hd
10 from emp
11 ) x
12 ) y

```

## SQL Server

使用函数 DATEDIFF，得到两个日期之间相差的月数（要得到相差年数，只需除以 12）：

```

1  select datediff(month,min_hd,max_hd),
2     datediff(month,min_hd,max_hd)/12
3  from (
4  select min(hiredate) min_hd, max(hiredate) max_hd
5  from emp
6  ) x

```

## 讨论

### DB2、MySQL 和 PostgreSQL

除 PostgreSQL 解决方案中从 MIN\_HD 和 MAX\_HD 提取了年份、月份的方法不同外，对于这 3 个 RDBM，计算 MIN\_HD 和 MAX\_HD 之间相差年数和月数的方法都相同。下面的讨论适用于这 3 种数据库的解决方案。内联视图 X 返回表 EMP 中第一个 HIREDATE 和最后一个 HIREDATE，如下所示：

```

select min(hiredate) as min_hd,
       max(hiredate) as max_hd
from emp

```

```

MIN_HD      MAX_HD
-----
17-DEC-1980 12-JAN-1983

```

要计算 MIN\_HD 和 MAX\_HD 之间的月数，只需用年数差乘以 12，然后再加上 MIN\_HD 和 MAX\_HD 之间的月数之差。如果不知道其中的机理，可以将这两个日期的有关部分显示出来。它们对年和月部分的数值如下所示：

```

select year(max_hd) as max_yr,   year(min_hd) as min_yr,
       month(max_hd) as max_mon, month(min_hd) as min_mon
from (

```

```
select min(hiredate) as min_hd, max(hiredate) as max_hd
from emp
) x
```

MAX_YR	MIN_YR	MAX_MON	MIN_MON
1983	1980	1	12

观察上面的结果, 会发现 MIN\_HD 和 MAX\_HD 之间相差的月数是  $(1983-1980)*12 + (1-12)$ 。要得到 MIN\_HD 和 MAX\_HD 之间相差的年数, 只需除以 12 即可, 当然, 还要根据用途, 对相差年数进行相应的舍 / 入操作。

## Oracle 和 SQL Server

内联视图 X 返回表 EMP 中第一个 HIREDATE 和最后一个 HIREDATE, 如下所示:

```
select min(hiredate) as min_hd, max(hiredate) as max_hd
from emp
```

MIN_HD	MAX_HD
17-DEC-1980	12-JAN-1983

由 Oracle 和 SQL Server 提供的函数 (分别为 MONTHS\_BETWEEN 和 DATEDIFF) 可以返回两个给定日期之间的月份数。要得到年数, 只需除以 12 即可。

## 8.5 确定两个日期之间的秒、分、小时数

### 问题

求两个日期之间相差的秒数, 例如, 求 ALLEN 和 WARD 的 HIREDATE (聘用日期) 之间相差的时间, 分别用秒、分、小时数表示。

### 解决方案

知道了两个日期之间的天数, 就可以计算出秒、分、小时数, 因为它们是组成一天的时间单位。

### DB2

使用函数 DAYS 获得 ALLEN\_HD 和 WARD\_HD 之间相差的天数。然后, 进行乘法操作, 就能得到每个时间单位的值:

```
1 select dy*24 hr, dy*24*60 min, dy*24*60*60 sec
2   from (
3 select ( days(max(case when ename = 'WARD'
4                  then hiredate
5                  end)) -
6         days(max(case when ename = 'ALLEN'
7                  then hiredate
8                  end))
9       ) as dy
10   from emp
11   ) x
```

## MySQL 和 SQL Server

使用函数 DATEDIFF 获得 ALLEN\_HD 和 WARD\_HD 之间相差的天数。然后，进行乘法操作，就能得到每个时间单位的值：

```

1 select datediff(day,allen_hd,ward_hd)*24 hr,
2        datediff(day,allen_hd,ward_hd)*24*60 min,
3        datediff(day,allen_hd,ward_hd)*24*60*60 sec
4   from (
5     select max(case when ename = 'WARD'
6                   then hiredate
7                   end) as ward_hd,
8            max(case when ename = 'ALLEN'
9                   then hiredate
10                  end) as allen_hd
11   from emp
12  ) x

```

## Oracle 和 PostgreSQL

使用减法操作，返回 ALLEN\_HD 和 WARD\_HD 之间相差的天数。然后，进行乘法操作，就能得到每个时间单位的值：

```

1 select dy*24 as hr, dy*24*60 as min, dy*24*60*60 as sec
2   from (
3     select (max(case when ename = 'WARD'
4                   then hiredate
5                   end) -
6            max(case when ename = 'ALLEN'
9                   then hiredate
10                  end)) as dy
7     from emp
8  ) x

```

## 讨论

所有解决方案的内联视图 X 都会返回 WARD 和 ALLEN 的 HIREDATE（聘用日期），如下所示：

```

select max(case when ename = 'WARD'
                then hiredate
                end) as ward_hd,
       max(case when ename = 'ALLEN'
                then hiredate
                end) as allen_hd
  from emp

```

WARD_HD	ALLEN_HD
22-FEB-1981	20-FEB-1981

将 WARD\_HD 和 ALLEN\_HD 之间相差的天数乘以 24（一天的小时数）、1440（一天的分钟数）、86400（一天的秒数）即可。

## 8.6 计算一年中周内各日期的次数

### 问题

计算一年中周内各日期（星期日、星期一……星期六）的次数。

## 解决方案

要计算一年中周内各日期分别有多少个，必须：

1. 生成一年内的所有日期。
2. 设置日期格式，得到每个日期对应为星期几。
3. 计数周内各日期分别有多少个。

### DB2

使用递归的 WITH 子句，以避免对至少包含 366 行的表进行 SELECT。使用函数 DAYNAME，确定每个日期为星期几，然后计数周内各日期的次数：

```

1 with x (start_date,end_date)
2 as (
3 select start_date,
4        start_date + 1 year end_date
5   from (
6 select (current_date -
7        dayofyear(current_date) day)
8        +1 day as start_date
9   from t1
10  ) tmp
11 union all
12 select start_date + 1 day, end_date
13   from x
14  where start_date + 1 day < end_date
15 )
16 select dayname(start_date),count(*)
17   from x
18  group by dayname(start_date)

```

### MySQL

对表 T500 进行选择操作，生成的行包含一年的所有日期。使用 DATE\_FORMAT 函数，确定每个日期为星期几，然后计数周内各日期的次数：

```

1 select date_format(
2         date_add(
3           cast(
4             concat(year(current_date),'-01-01')
5               as date),
6             interval t500.id-1 day),
7           '%W') day,
8         count(*)
9   from t500
10  where t500.id <= datediff(
11         cast(
12           concat(year(current_date)+1,'-01-01')
13             as date),
14         cast(
15           concat(year(current_date),'-01-01')
16             as date))
17  group by date_format(
18         date_add(
19           cast(
20             concat(year(current_date),'-01-01')
21               as date),
22             interval t500.id-1 day),

```



23

'%W')

## Oracle

对于 Oracle9i Database 或更高版本, 可以使用递归的 CONNECT BY 子句, 返回一年内的所有日期。而对于 Oracle8i Database 或较早版本, 则只需对表 T500 进行选择操作, 就能生成包含一年内所有日期的行。另外, 使用 TO\_CHAR 函数, 可确定每个日期为星期几, 然后计数周内各日期的次数:

首先是 CONNECT BY 解决方案:

```

1 with x as (
2   select level lvl
3     from dual
4   connect by level <= (
5     add_months(trunc(sysdate,'y'),12)-trunc(sysdate,'y')
6   )
7 )
8 select to_char(trunc(sysdate,'y')+lvl-1,'DAY'), count(*)
9   from x
10  group by to_char(trunc(sysdate,'y')+lvl-1,'DAY')

```

接下来是 Oracle 较早版本的解决方案:

```

1 select to_char(trunc(sysdate,'y')+rownum-1,'DAY'),
2        count(*)
3   from t500
4  where rownum <= (add_months(trunc(sysdate,'y'),12)
5                    - trunc(sysdate,'y'))
6  group by to_char(trunc(sysdate,'y')+rownum-1,'DAY')

```

## PostgreSQL

使用内置函数 GENERATE\_SERIES, 生成包含一年内所有日期的行。然后使用 TO\_CHAR 函数确定每个日期为星期几, 然后计数周内各日期的次数。例如:

```

1 select to_char(
2   cast(
3     date_trunc('year',current_date)
4     as date) + gs.id-1,'DAY'),
5        count(*)
6   from generate_series(1,366) gs(id)
7  where gs.id <= (cast
8    ( date_trunc('year',current_date) +
9      interval '12 month' as date) -
10   cast(date_trunc('year',current_date)
11     as date))
12  group by to_char(
13    cast(
14      date_trunc('year',current_date)
15      as date) + gs.id-1,'DAY')

```

## SQL Server

使用递归的 WITH 子句, 以避免对至少包含 366 行的表进行 SELECT。在不支持 WITH

子句的 SQL Server 版本中，需要使用基于表，有关内容可以参考 Oracle 的第二个解决方案。使用函数 DAYNAME，可确定每个日期为星期几，然后计数周内各日期的次数。例如：

```

1  with x (start_date,end_date)
2  as (
3  select start_date,
4         dateadd(year,1,start_date) end_date
5  from (
6  select cast(
7         cast(year(getdate( )) as varchar) + '-01-01'
8         as datetime) start_date
9  from t1
10 ) tmp
11 union all
12 select dateadd(day,1,start_date), end_date
13 from x
14 where dateadd(day,1,start_date) < end_date
15 )
16 select datename(dw,start_date),count(*)
17 from x
18 group by datename(dw,start_date)
19 OPTION (MAXRECURSION 366)

```

## 讨论

### DB2

递归 WITH 视图 X 的内联视图 TMP 将返回当前年份的第一天，如下所示：

```

select (current_date -
        dayofyear(current_date) day)
       +1 day as start_date
from t1

START_DATE
-----
01-JAN-2005

```

下一步，给 START\_DATE 加 1 年，这样就有了开始日期和结束日期。想要生成一年内的每一天，就必须知道这两个日期。START\_DATE 和 END\_DATE 如下所示：

```

select start_date,
       start_date + 1 year end_date
from (
select (current_date -
        dayofyear(current_date) day)
       +1 day as start_date
from t1
) tmp

START_DATE  END_DATE
-----
01-JAN-2005 01-JAN-2006

```

接下来，递归增加 START\_DATE，每次加 1 天，直到它等于 END\_DATE 时为止。下面列出了由递归视图 X 返回的部分行：

```

with x (start_date,end_date)
as (
select start_date,
       start_date + 1 year end_date
from (

```

```

select (current_date -
        dayofyear(current_date) day)
       +1 day as start_date
  from t1
     ) tmp
 union all
select start_date + 1 day, end_date
  from x
 where start_date + 1 day < end_date
 )
select * from x

```

START_DATE	END_DATE
01-JAN-2005	01-JAN-2006
02-JAN-2005	01-JAN-2006
03-JAN-2005	01-JAN-2006
...	
29-JAN-2005	01-JAN-2006
30-JAN-2005	01-JAN-2006
31-JAN-2005	01-JAN-2006
...	
01-DEC-2005	01-JAN-2006
02-DEC-2005	01-JAN-2006
03-DEC-2005	01-JAN-2006
...	
29-DEC-2005	01-JAN-2006
30-DEC-2005	01-JAN-2006
31-DEC-2005	01-JAN-2006

最后一步，对于由递归视图 X 返回的行，使用函数 DAYNAME，然后计数周内各日期的次数。下面列出了其最后结果：

```

with x (start_date,end_date)
as (
select start_date,
       start_date + 1 year end_date
  from (
select (current_date -
        dayofyear(current_date) day)
       +1 day as start_date
  from t1
     ) tmp
 union all
select start_date + 1 day, end_date
  from x
 where start_date + 1 day < end_date
 )
select dayname(start_date),count(*)
  from x
 group by dayname(start_date)

```

START_DATE	COUNT(*)
FRIDAY	52
MONDAY	52
SATURDAY	53
SUNDAY	52
THURSDAY	52
TUESDAY	52
WEDNESDAY	52

## MySQL

该解决方案对表 T500 进行选择，以便生成包含一年内所有日期的行。第 4 行的命令返回

当前年份的第一天，其方法是求出函数 CURRENT\_DATE 所返回日期的年份，并添加月份和天（采用 MySQL 的默认数据格式），其结果如下：

```
select concat(year(current_date), '-01-01')
from t1

START_DATE
-----
01-JAN-2005
```

得到了当前年份的第一天以后，则可使用 DATEADD 函数在该日期上分别加上 T500.ID 中的每个值，以生成该年份的每一天。使用函数 DATE\_FORMAT，可返回每个日期是星期几。要从表 T500 中选择想要的行数，只需计算出当前年份的第一天与下一个年份的第一天之间的日期差，并返回相应数目的行（365 或 366 个）。下面列出了部分结果：

```
select date_format(
    date_add(
        cast(
            concat(year(current_date), '-01-01')
            as date),
        interval t500.id-1 day),
    '%W') day
from t500
where t500.id <= datediff(
    cast(
        concat(year(current_date)+1, '-01-01')
        as date),
    cast(
        concat(year(current_date), '-01-01')
        as date))

DAY
-----
01-JAN-2005
02-JAN-2005
03-JAN-2005
...
29-JAN-2005
30-JAN-2005
31-JAN-2005
...
01-DEC-2005
02-DEC-2005
03-DEC-2005
...
29-DEC-2005
30-DEC-2005
31-DEC-2005
```

得到了当前年份的每一天后，则可以计数由函数 DAYNAME 返回的周内各日期各自的出现次数。最终结果如下所示：

```
select date_format(
    date_add(
        cast(
            concat(year(current_date), '-01-01')
            as date),
        interval t500.id-1 day),
    '%W') day,
    count(*)
from t500
where t500.id <= datediff(
```

```

        cast(
        concat(year(current_date)+1,'-01-01')
              as date),
        cast(
        concat(year(current_date),'-01-01')
              as date))
group by date_format(
        date_add(
        cast(
        concat(year(current_date),'-01-01')
              as date),
        interval t500.id-1 day),
        '%W')

```

DAY	COUNT(*)
FRIDAY	52
MONDAY	52
SATURDAY	53
SUNDAY	52
THURSDAY	52
TUESDAY	52
WEDNESDAY	52

## Oracle

这个解决方案既提供了对表 T500（基干表）的选择操作的方法，也提供了使用递归的 CONNECT BY 和 WITH 子句的方法，为当前年的每一天生成一行信息。调用函数 TRUNC，可以从当前日期截取当前年份的第一天。

如果采用 CONNECT BY/WITH 解决方案，则可以使用伪列 (pseudo-column) LEVEL 生成起始值为 1 的序列数。要生成这种解决方案所要求数目的行，需根据当前年的第一天与下一年的第一天之间相差的天数（365 或 366 个）对 ROWNUM 或 LEVEL 进行筛选；然后，通过给当前年的第一天加上 ROWNUM 或 LEVEL，递增得到一年中的每一天。下面列出了部分结果：

```

/* Oracle 9i and later */
with x as (
select level lvl
  from dual
 connect by level <= (
    add_months(trunc(sysdate,'y'),12)-trunc(sysdate,'y')
  )
)
select trunc(sysdate,'y')+lvl-1
  from x

```

如果采用基干表解决方案，则可以在其内部使用至少包含 366 行的任意表或视图。因为 Oracle 引入了函数 ROWNUM，所以表中不需要包含从 1 开始递增的数值。参阅下面的例子，它使用基干表 T500 返回当前年的每一天：

```

/* Oracle 8i and earlier */
select trunc(sysdate,'y')+rownum-1 start_date
  from t500
 where rownum <= (add_months(trunc(sysdate,'y'),12)
                  - trunc(sysdate,'y'))

```

```

START_DATE
-----
01-JAN-2005

```

```

02-JAN-2005
03-JAN-2005
...
29-JAN-2005
30-JAN-2005
31-JAN-2005
...
01-DEC-2005
02-DEC-2005
03-DEC-2005
...
29-DEC-2005
30-DEC-2005
31-DEC-2005

```

无论采用哪种方法，最终都要使用函数 TO\_CHAR 返回每个日期是星期几，并计算各自出现的次数。其结果如下所示：

```

/* Oracle 9i and later */
with x as (
select level lvl
  from dual
 connect by level <= (
    add_months(trunc(sysdate,'y'),12)-trunc(sysdate,'y')
  )
)
select to_char(trunc(sysdate,'y')+lvl-1,'DAY'), count(*)
  from x
 group by to_char(trunc(sysdate,'y')+lvl-1,'DAY')
/* Oracle 8i and earlier */
select to_char(trunc(sysdate,'y')+rownum-1,'DAY') start_date,
       count(*)
  from t500
 where rownum <= (add_months(trunc(sysdate,'y'),12)
                  - trunc(sysdate,'y'))
 group by to_char(trunc(sysdate,'y')+rownum-1,'DAY')

```

START_DATE	COUNT(*)
FRIDAY	52
MONDAY	52
SATURDAY	53
SUNDAY	52
THURSDAY	52
TUESDAY	52
WEDNESDAY	52

## PostgreSQL

首先使用 DATE\_TRUNC 函数返回当前日期的年份（如下所示，对 T1 进行选择操作，只返回一行信息）：

```

select cast(
  date_trunc('year',current_date)
  as date) as start_date
  from t1

```

START_DATE
01-JAN-2005

然后，对至少包含 366 行的一个行来源（实际上任何表表达式均可）进行选择操作。本解决方案把函数 GENERATE\_SERIES 当作行来源使用，当然，也可以使用表 T500。然

后在当前年的第一天上加上各行的序号（原文是加1，不妥，译者注），直到返回一年内的每一天（如下所示）：

```
select cast( date_trunc('year',current_date)
            as date) + gs.id-1 as start_date
  from generate_series (1,366) gs(id)
 where gs.id <= (cast
                ( date_trunc('year',current_date) +
                  interval '12 month' as date) -
            cast(date_trunc('year',current_date)
                as date))

START_DATE
-----
01-JAN-2005
02-JAN-2005
03-JAN-2005
...
29-JAN-2005
30-JAN-2005
31-JAN-2005
...
01-DEC-2005
02-DEC-2005
03-DEC-2005
...
29-DEC-2005
30-DEC-2005
31-DEC-2005
```

最后，使用函数 TO\_CHAR 返回每个日期是星期几，再计数周内各日期的次数。最终结果如下所示：

```
select to_char(
        cast(
            date_trunc('year',current_date)
            as date) + gs.id-1,'DAY') as start_dates,
       count(*)
  from generate_series(1,366) gs(id)
 where gs.id <= (cast
                ( date_trunc('year',current_date) +
                  interval '12 month' as date) -
            cast(date_trunc('year',current_date)
                as date))
 group by to_char(
        cast(
            date_trunc('year',current_date)
            as date) + gs.id-1,'DAY')

START_DATE  COUNT(*)
-----
FRIDAY      52
MONDAY      52
SATURDAY    53
SUNDAY      52
THURSDAY    52
TUESDAY     52
WEDNESDAY   52
```

## SQL Server

递归 WITH 视图 X 中的内联视图 TMP 返回当前年份的第一天，如下所示：

```
select cast(
```

```

        cast(year(getdate( )) as varchar) + '-01-01'
        as datetime) start_date
    from t1
START_DATE
-----
01-JAN-2005

```

在有了当前年份的第一天后，给 START\_DATE 加1年，这样既有开始日期，又有结束日期。要生成一年内的每一天，必须知道这两个值。START\_DATE 和 END\_DATE 如下所示：

```

select start_date,
       dateadd(year,1,start_date) end_date
  from (
select cast(
       cast(year(getdate( )) as varchar) + '-01-01'
       as datetime) start_date
  from t1
    ) tmp
START_DATE  END_DATE
-----
01-JAN-2005 01-JAN-2006

```

接下来，递归递增 START\_DATE 值，每次加1，直到它等于 END\_DATE 为止（不包括 END\_DATE）。下面给出了由递归视图 X 返回的部分行：

```

with x (start_date,end_date)
as (
select start_date,
       dateadd(year,1,start_date) end_date
  from (
select cast(
       cast(year(getdate( )) as varchar) + '-01-01'
       as datetime) start_date
  from t1
    ) tmp
union all
select dateadd(day,1,start_date), end_date
  from x
 where dateadd(day,1,start_date) < end_date
)
select * from x
OPTION (MAXRECURSION 366)
START_DATE  END_DATE
-----
01-JAN-2005 01-JAN-2006
02-JAN-2005 01-JAN-2006
03-JAN-2005 01-JAN-2006
...
29-JAN-2005 01-JAN-2006
30-JAN-2005 01-JAN-2006
31-JAN-2005 01-JAN-2006
...
01-DEC-2005 01-JAN-2006
02-DEC-2005 01-JAN-2006
03-DEC-2005 01-JAN-2006
...
29-DEC-2005 01-JAN-2006
30-DEC-2005 01-JAN-2006
31-DEC-2005 01-JAN-2006

```



最后, 对递归视图X返回的行使用函数DATENAME, 并计数周内各日期的出现次数, 其结果如下:

```
with x(start_date,end_date)
as (
  select start_date,
         dateadd(year,1,start_date) end_date
  from (
    select cast(
      cast(year(getdate( )) as varchar) + '-01-01'
      as datetime) start_date
    from t1
  ) tmp
union all
select dateadd(day,1,start_date), end_date
  from x
 where dateadd(day,1,start_date) < end_date
)
select datename(dw,start_date), count(*)
  from x
 group by datename(dw,start_date)
OPTION (MAXRECURSION 366)
```

START_DATE	COUNT(*)
FRIDAY	52
MONDAY	52
SATURDAY	53
SUNDAY	52
THURSDAY	52
TUESDAY	52
WEDNESDAY	52

## 8.7 确定当前记录和下一条记录之间相差的天数

### 问题

求两个日期(指存储在两个不同行内的日期)之间相差的天数。例如, 对于DEPTNO 10中的每个员工, 确定聘用他们的日期及聘用下一个员工(可能是其他部门的员工)的日期之间相差的天数。

### 解决方案

这个问题的解决方案是找到当前员工聘用的最早HIREDATE(聘用日期)。然后, 只需采用8.2节中介绍的技巧, 就能够得到相差天数。

#### DB2

使用标量子查询, 可以找到当前HIREDATE的下一个HIREDATE。然后, 使用DAYS函数获得相差天数:

```
1 select x.*,
2       days(x.next_hd) - days(x.hiredate) diff
3   from (
4   select e.deptno, e.ename, e.hiredate,
5          (select min(d.hiredate) from emp d
6           where d.hiredate > e.hiredate) next_hd
7   from emp e
```

```

8   where e.deptno = 10
9   ) x

```

## MySQL 和 SQL Server

使用标量子查询,可以找到当前HIREDATE的下一个HIREDATE。然后,使用DATEDIFF函数获得相差天数。SQL Server 版本的 DATEDIFF 的用法如下:

```

1  select x.*,
2     datediff(day,x.hiredate,x.next_hd) diff
3  from (
4  select e.deptno, e.ename, e.hiredate,
5     (select min(d.hiredate) from emp d
6      where d.hiredate > e.hiredate) next_hd
7  from emp e
8  where e.deptno = 10
9  ) x

```

MySQL 用户去掉第一个参数,并交换两个参数的顺序:

```

2     datediff(x.next_hd, x.hiredate) diff

```

## Oracle

在 Oracle8i Database 或更高的版本中,需使用窗口函数 LEAD OVER 访问当前行的下一个 HIREDATE,进而进行减法操作:

```

1  select ename, hiredate, next_hd,
2     next_hd - hiredate diff
3  from (
4  select deptno, ename, hiredate,
5     lead(hiredate)over(order by hiredate) next_hd
6  from emp
7  )
8  where deptno=10

```

在 Oracle8 Database 或较早版本中,可以采用 PostgreSQL 解决方案。

## PostgreSQL

使用标量子查询,可以找到当前HIREDATE的下一个HIREDATE。然后,只需用减法即可获得相差天数:

```

1  select x.*,
2     x.next_hd - x.hiredate as diff
3  from (
4  select e.deptno, e.ename, e.hiredate,
5     (select min(d.hiredate) from emp d
6      where d.hiredate > e.hiredate) as next_hd
7  from emp e
8  where e.deptno = 10
9  ) x

```

## 讨论

### DB2、MySQL、PostgreSQL 和 SQL Server

对于以上这些解决方案,尽管其语法不同,但步骤是相同的:使用标量子查询,找到当

前 HIREDATE 的下一个 HIREDATE，然后采用本章前面 8.2 节中介绍的技巧，获得两者之间相差的天数。

## Oracle

这里提到的窗口函数 LEAD OVER 非常有用，它能够访问“未来的”行（“未来的”行相对于当前行，由 ORDER BY 子句决定）。这种无需添加联接就能够访问当前行附近行的功能，提高了代码的可读性和有效性。在采用窗口函数时，一定要记住，它在 WHERE 子句之后求值，因此在该解决方案中，需要使用内联视图。如果把对 DEPTNO 的筛选移到内联视图，则结果会发生改变（仅考虑了 DEPTNO 10 中的 HIREDATE）。对于 Oracle 的 LEAD 和 LAG 函数还需要特别注意，它们的结果中可能会有重复。本书前言中曾经提到过，各章节的代码中没有采取“防御性的”措施，这是因为条件太多了，不可能预见到所有可能中断代码执行的情况；即使能够预见所有问题，但其结果可能是 SQL 语句难以理解。因此，在大多数情况下，解决方案的目标是介绍一种技巧：可以在产品系统中使用它，但一定要依据特定的数据反复进行测试和改进。尽管如此，在这个特定的例子中还是要对此稍加讨论，只因其处理方式不那么显而易见，对非 Oracle 系统的用户来说尤其如此。在这个例子中，表 EMP 内不包含重复的 HIREDATE，但是，在其他表中，很可能有重复的日期值。下面给出了 DEPTNO 10 中的员工及他们的 HIREDATE：

```
select ename, hiredate
  from emp
 where deptno=10
 order by 2

ENAME    HIREDATE
-----
CLARK    09-JUN-1981
KING     17-NOV-1981
MILLER   23-JAN-1982
```

为了讲清楚这个例子，向表中插入 4 个重复值，这样在 11 月 17 日聘用了 5 个员工（其中包括 KING）：

```
insert into emp (empno,ename,deptno,hiredate)
values (1,'ant',10,to_date('17-NOV-1981'))

insert into emp (empno,ename,deptno,hiredate)
values (2,'joe',10,to_date('17-NOV-1981'))

insert into emp (empno,ename,deptno,hiredate)
values (3,'jim',10,to_date('17-NOV-1981'))

insert into emp (empno,ename,deptno,hiredate)
values (4,'choi',10,to_date('17-NOV-1981'))

select ename, hiredate
  from emp
 where deptno=10
 order by 2

ENAME    HIREDATE
-----
CLARK    09-JUN-1981
ant      17-NOV-1981
joe      17-NOV-1981
```

```

KING      17-NOV-1981
jim       17-NOV-1981
choi      17-NOV-1981
MILLER    23-JAN-1982

```

目前, DEPTNO 10同一天聘用了多个员工。如果试图对此结果集使用前面提到的解决方案(即把筛选移到内联视图, 这样仅考虑DEPTNO 10的员工及他们的HIREDATE), 将得到如下输出:

```

select ename, hiredate, next_hd,
       next_hd - hiredate diff
  from (
select deptno, ename, hiredate,
       lead(hiredate)over(order by hiredate) next_hd
  from emp
 where deptno=10
 )

```

ENAME	HIREDATE	NEXT_HD	DIFF
CLARK	09-JUN-1981	17-NOV-1981	161
ant	17-NOV-1981	17-NOV-1981	0
joe	17-NOV-1981	17-NOV-1981	0
KING	17-NOV-1981	17-NOV-1981	0
jim	17-NOV-1981	17-NOV-1981	0
choi	17-NOV-1981	23-JAN-1982	67
MILLER	23-JAN-1982	(null)	(null)

查看一下同一天聘用的5个员工的DIFF值就会发现, 其中4个人为0, 这是错误的。同一天聘用的所有员工都应该跟下一个聘用其他员工的HIREDATE进行计算, 即17日聘用的所有员工, 都应该跟MILLER的HIREDATE进行计算。这里的问题是: LEAD函数按HIREDATE对行排序, 而且它不会跳过重复行, 所以以ANT为例, 若将ANT的HIREDATE跟员工JOE的HIREDATE进行计算, 则差为0, 因此ANT的DIFF值为0。所幸Oracle针对这类情况提供了一个非常简单的应对措施: 当调用LEAD函数时, 可以给LEAD传递一个参数, 以便准确地指定“未来的”行(是下一行? 10行之后? 等等)所在。所以, 再看看员工ANT的情况, 不是要向前查一行, 而需要向前查5行(需要跳过其他所有重复行), 这里是MILLER。如果查看一下员工JOE, 他距离MILLER有4行, JIM距离MILLER3行, KING距离MILLER2行, 帅哥CHOI距离MILLER只有1行。要得到正确答案, 只需把每个员工距MILLER的行数作为参数传递给LEAD。其解决方案如下所示:

```

select ename, hiredate, next_hd,
       next_hd - hiredate diff
  from (
select deptno, ename, hiredate,
       lead(hiredate,cnt-rn+1)over(order by hiredate) next_hd
  from (
select deptno,ename,hiredate,
       count(*)over(partition by hiredate) cnt,
       row_number( )over(partition by hiredate order by empno) rn
  from emp
 where deptno=10
 )
 )

```

ENAME	HIREDATE	NEXT_HD	DIFF
CLARK	09-JUN-1981	17-NOV-1981	161
ant	17-NOV-1981	23-JAN-1982	67
joe	17-NOV-1981	23-JAN-1982	67
jim	17-NOV-1981	23-JAN-1982	67
choi	17-NOV-1981	23-JAN-1982	67
KING	17-NOV-1981	23-JAN-1982	67
MILLER	23-JAN-1982	(null)	(null)

现在，结果对了。同一天聘用的所有员工都有自己的 HIREDATE，它们是相对于下一个 HIREDATE（而不是与自己相同的 HIREDATE）计算出来的。如果觉得该对应措施不那么清楚，只需将该查询拆开，先从内联视图开始：

```
select deptno,ename,hiredate,
       count(*)over(partition by hiredate) cnt,
       row_number( )over(partition by hiredate order by empno) rn
from emp
where deptno=10
```

DEPTNO	ENAME	HIREDATE	CNT	RN
10	CLARK	09-JUN-1981	1	1
10	ant	17-NOV-1981	5	1
10	joe	17-NOV-1981	5	2
10	jim	17-NOV-1981	5	3
10	choi	17-NOV-1981	5	4
10	KING	17-NOV-1981	5	5
10	MILLER	23-JAN-1982	1	1

窗口函数 COUNT OVER 用于计算每个 HIREDATE 重复的次数，并返回给每一行。对于重复的 HIREDATE，该 HIREDATE 的每一行都返回 5；窗口函数 ROW\_NUMBER OVER 按照 EMPNO 给每个员工制定序号并且序号依照 HIREDATE 分区，因此如果没有重复的 HIREDATE，则每个员工的序号都为 1。至此，所有重复日期都做了计数并制定了序号，可以用该序号来表示距下一个 HIREDATE（MILLER 的 HIREDATE）的行数。当调用 LEAD 时，将每行的 CNT 减 RN 加 1 作为参数，就可以看到如下结果：

```
select deptno, ename, hiredate,
       cnt-rn+1 distance_to_miller,
       lead(hiredate,cnt-rn+1)over(order by hiredate) next_hd
from (
select deptno,ename,hiredate,
       count(*)over(partition by hiredate) cnt,
       row_number( )over(partition by hiredate order by empno) rn
from emp
where deptno=10
)
```

DEPTNO	ENAME	HIREDATE	DISTANCE_TO_MILLER	NEXT_HD
10	CLARK	09-JUN-1981	1	17-NOV-1981
10	ant	17-NOV-1981	5	23-JAN-1982
10	joe	17-NOV-1981	4	23-JAN-1982
10	jim	17-NOV-1981	3	23-JAN-1982
10	choi	17-NOV-1981	2	23-JAN-1982
10	KING	17-NOV-1981	1	23-JAN-1982
10	MILLER	23-JAN-1982	1	(null)

可以看到，传递给它向前跳越的距离值后，LEAD 函数就可以相对于正确的日期进行减法操作了。

## 第 9 章

# 日期操作

本章将介绍查找和修改日期的方法。日期的查询操作是很常见的，因此，一定要知道如何处理日期，而且必须熟悉 RDBMS 平台处理日期的函数。本章内容是后面进行其他复杂查询操作的重要基础，不仅涉及到日期，还有时间。

在了解这些内容之前，先强调一下（前言中曾提到过），以这些解决方案为指南，可以解决特殊问题。要学会举一反三，触类旁通。例如，如果某个方案解决了当前月份的问题，那么要记住不仅仅是这个解决方案中的月份，其他月份也可以使用该方案（只需很少的修改）。另外，把这些方案作为指南，而不是作为最终的选择。一本书不会容纳所有问题的答案，但是，如果理解了本书介绍的内容，那么，修改这些解决方案来满足自己的需要是相当容易的。读者也应该考虑本书提出的其他解决方案。例如，如果使用了 RDBMS 提供的特殊函数解决某个问题，那么，就值得花费一些时间和精力，去探究是否有其他解决方案——有的可能更好一些，有的可能稍差点。了解多种解决方案，将有助于读者成为更好的 SQL 程序员。

---

注意：本章提到的方案都采用了简单的日期类型。对于更复杂的日期类型，应该调整解决方案。

---

### 9.1 确定一年是否为闰年问题

确定当前年是否为闰年。

#### 解决方案

如果曾经用过一段时间 SQL，就应当已经知道了解决这种问题的技巧。所有解决方案都很好用，但本节采用了最简单的方案。该解决方案只是检查 2 月的最后一天，如果它为 29，则当前年就为闰年。

## DB2

采用递归的 WITH 子句，可返回 2 月份的每一天。使用聚集函数 MAX 确定 2 月的最后一天。

```

1  with x (dy,mth)
2  as (
3  select dy, month(dy)
4  from (
5  select (current_date -
6         dayofyear(current_date) days +1 days)
7         +1 months as dy
8  from t1
9  ) tmp1
10 union all
11 select dy+1 days, mth
12 from x
13 where month(dy+1 day) = mth
14 )
15 select max(day(dy))
16 from x

```

## Oracle

使用函数 LAST\_DAY，可计算出 2 月份的最后一天：

```

1 select to_char(
2         last_day(add_months(trunc(sysdate,'y'),1)),
3         'DD')
4 from t1

```

## PostgreSQL

使用函数 GENERATE\_SERIES，可返回 2 月份的每一天。使用聚集函数 MAX 确定 2 月的最后一天：

```

1 select max(to_char(tmp2.dy+x.id,'DD')) as dy
2 from (
3 select dy, to_char(dy,'MM') as mth
4 from (
5 select cast(cast(
6         date_trunc('year',current_date) as date)
7         + interval '1 month' as date) as dy
8 from t1
9 ) tmp1
10 ) tmp2, generate_series (0,29) x(id)
11 where to_char(tmp2.dy+x.id,'MM') = tmp2.mth

```

## MySQL

使用函数 LAST\_DAY 获得 2 月的最后一天：

```

1 select day(
2         last_day(
3         date_add(
4         date_add(
5         date_add(current_date,
6                 interval -dayofyear(current_date) day),
7                 interval 1 day),
8                 interval 1 month))) dy
9 from t1

```

## SQL Server

采用递归 WITH 子句, 可返回 2 月份的每一天。使用聚集函数 MAX 确定 2 月的最后一天:

```

1  with x (dy,mth)
2  as (
3  select dy, month(dy)
4  from (
5  select dateadd(mm,1,(getdate()-datepart(dy,getdate()))+1) dy
6  from t1
7  ) tmp1
8  union all
9  select dateadd(dd,1,dy), mth
10 from x
11 where month(dateadd(dd,1,dy)) = mth
12 )
13 select max(day(dy))
14 from x

```

## 讨论

### DB2

递归视图 X 中的内联视图 TMP1 将返回 2 月份的每一天, 步骤如下:

1. 获得当前日期。
2. 使用 DAYOFYEAR, 确定当前日期的年份日期 1。
3. 从当前日期中减去第 2 步计算出来的值, 以得到前一年的 12 月 31 日, 然后加 1, 就获得了当前年的 1 月 1 日。
4. 再加 1 个月, 就得到 2 月 1 日。

这种算法的结果如下:

```

select (current_date -
        dayofyear(current_date) days +1 days) +1 months as dy
from t1

DY
-----
01-FEB-2005

```

然后, 使用 MONTH 函数, 返回由内联视图 TMP1 获得的日期的月份:

```

select dy, month(dy) as mth
from (
select (current_date -
        dayofyear(current_date) days +1 days) +1 months as dy
from t1
) tmp1

DY          MTH
-----
01-FEB-2005    2

```

目前得到的结果, 为获得 2 月每一天的递归操作提供了起始条件。要返回 2 月的每一天, 可给 DY 重复加 1, 直到进入 3 月份为止。下面给出了 WITH 操作的部分结果:



```

with x (dy,mth)
as (
select dy, month(dy)
from (
select (current_date -
       dayofyear(current_date) days +1 days) +1 months as dy
from t1
) tmp1
union all
select dy+1 days, mth
from x
where month(dy+1 day) = mth
)
select dy,mth
from x

```

DY	MTH
01-FEB-2005	2
...	
10-FEB-2005	2
...	
28-FEB-2005	2

最后一步是对 DY 列使用 MAX 函数，以返回 2 月的最后一天；如果它为 29，则为闰年。

## Oracle

首先，使用 TRUNC 函数找到一年的第一天：

```

select trunc(sysdate,'y')
from t1

```

DY
01-JAN-2005

因为一年的第一天是 1 月 1 日，所以下一步是加 1 个月，得到 2 月 1 日：

```

select add_months(trunc(sysdate,'y'),1) dy
from t1

```

DY
01-FEB-2005

然后，使用 LAST\_DAY 函数获得 2 月的最后一天：

```

select last_day(add_months(trunc(sysdate,'y'),1)) dy
from t1

```

DY
28-FEB-2005

最后一步（可选）是使用 TO\_CHAR 返回 28 或 29。

## PostgreSQL

首先，检查由内联视图 TMP1 返回的结果。使用 DATE\_TRUNC 函数获得当前年的第一天，并把结果转换为 DATE 格式：

```

select cast(date_trunc('year',current_date) as date) as dy

```

```

from t1
DY
-----
01-JAN-2005

```

然后，给当前年的第一天加一个月，以获得2月的第一天，并把结果转换为日期格式：

```

select cast(cast(
            date_trunc('year',current_date) as date)
            + interval '1 month' as date) as dy
from t1
DY
-----
01-FEB-2005

```

下一步，从内联视图 TMP1 中返回 DY 以及 DY 的数字月份值。使用 TO\_CHAR 函数，可返回数字月份值，如下所示：

```

select dy, to_char(dy,'MM') as mth
from (
select cast(cast(
            date_trunc('year',current_date) as date)
            + interval '1 month' as date) as dy
from t1
) tmp1
DY          MTH
-----
01-FEB-2005 2

```

至此所得到的就是内联视图 TMP2 的结果集。下一步使用极其有用的函数 GENERATE\_SERIES 返回 29 行（从 1~29）信息。分别将由 GENERATE\_SERIES（别名 X）返回的每一行的值跟内联视图 TMP2 的 DY 相加，下面列出了部分结果：

```

select tmp2.dy+x.id as dy, tmp2.mth
from (
select dy, to_char(dy,'MM') as mth
from (
select cast(cast(
            date_trunc('year',current_date) as date)
            + interval '1 month' as date) as dy
from t1
) tmp1
) tmp2, generate_series (0,29) x(id)
where to_char(tmp2.dy+x.id,'MM') = tmp2.mth
DY          MTH
-----
01-FEB-2005 02
...
10-FEB-2005 02
...
28-FEB-2005 02

```

最后一步是使用 MAX 函数获得 2 月的最后一天。再使用函数 TO\_CHAR 返回 28 或 29。

## MySQL

首先查找当前年的第一天，先从当前日期中减掉它的年份日期，然后加 1 天。使用 DATE\_ADD 函数可实现此功能：

```

select date_add(
    date_add(current_date,
        interval -dayofyear(current_date) day),
        interval 1 day) dy
from t1
DY
-----
01-JAN-2005

```

下一步，使用 DATE\_ADD 函数加一个月：

```

select date_add(
    date_add(
        date_add(current_date,
            interval -dayofyear(current_date) day),
            interval 1 day),
        interval 1 month) dy
from t1
DY
-----
01-FEB-2005

```

至此，进入了 2 月份，使用 LAST\_DAY 函数可得到当前月的最后一天：

```

select last_day(
    date_add(
        date_add(
            date_add(current_date,
                interval -dayofyear(current_date) day),
                interval 1 day),
            interval 1 month)) dy
from t1
DY
-----
28-FEB-2005

```

最后一步（可选）是使用 DAY 函数返回 28 或 29。

## SQL Server

该解决方案将使用递归 WITH 子句生成 2 月的每一天。首先找到 2 月的第一天。要找到这一天，需获得当前年的第一天，先从当前日期中减掉它的年份日期，然后加 1 天。一旦知道了当前年的第一天，就可使用 DATEADD 函数加 1 个月，以便进入 2 月第一天：

```

select dateadd(mm,1,(getdate()-datepart(dy,getdate()))+1) dy
from t1
DY
-----
01-FEB-2005

```

然后，返回 2 月的第一天以及 2 月的数字月份值：

```

select dy, month(dy) mth
from (
    select dateadd(mm,1,(getdate()-datepart(dy,getdate()))+1) dy
    from t1
    ) tmp1
DY          MTH
-----
01-FEB-2005  2

```

下一步, 使用 WITH 子句的递归功能, 给内联视图 TMP1 的 DY 重复加 1 天, 直到进入 3 月份为止 (不包括 3 月份的日期), 下面给出了部分结果:

```
with x (dy,mth)
as (
select dy, month(dy)
from (
select dateadd(mm,1,(getdate()-datepart(dy,getdate()))+1) dy
from t1
) tmp1
union all
select dateadd(dd,1,dy), mth
from x
where month(dateadd(dd,1,dy)) = mth
)
select dy,mth from x
```

DY	MTH
01-FEB-2005	02
...	
10-FEB-2005	02
...	
28-FEB-2005	02

至此, 已可以返回 2 月的每一天, 然后, 使用 MAX 函数查看最后一天是 28 日还是 29 日。最后一步也可以不用日期比较, 而使用 DAY 函数返回 28 或 29。

## 9.2 确定一年内的天数

### 问题

计算当前年的天数。

### 解决方案

当前年的天数等于第二年的第一天与当前年第一天 (以日为单位) 之差。针对每个系统的解决方案, 以下步骤都相同:

1. 找到当前年的第一天。
2. 给该日期加 1 年 (即可得到第二年的第一天)。
3. 从第二步的结果中减去当前年。

各解决方案的区别仅在于完成上述步骤时所使用的内置函数不同。

### DB2

使用函数 DAYOFYEAR 定位当前年的第一天, 并使用 DAYS 获得当前年的天数:

```
1 select days((curr_year + 1 year)) - days(curr_year)
2   from (
3 select (current_date -
```

```

4         dayofyear(current_date) day +
5         1 day) curr_year
6   from t1
7   ) x

```

## Oracle

使用函数TRUNC，找到当前年的第一天，并使用ADD\_MONTHS获得下一年的第一天：

```

1 select add_months(trunc(sysdate,'y'),12) - trunc(sysdate,'y')
2   from dual

```

## PostgreSQL

使用函数DATE\_TRUNC，找到当前年的第一天。然后使用间隔算法确定下一年的第一天：

```

1 select cast((curr_year + interval '1 year') as date) - curr_year
2   from (
3 select cast(date_trunc('year',current_date) as date) as curr_year
4   from t1
5   ) x

```

## MySQL

使用ADDDATE，找到当前年的第一天。再使用DATEDIFF及间隔算法确定当前年的天数：

```

1 select datediff((curr_year + interval 1 year),curr_year)
2   from (
3 select adddate(current_date,-dayofyear(current_date)+1) curr_year
4   from t1
5   ) x

```

## SQL Server

使用函数DATEADD得到当前年的第一天。使用DATEDIFF返回当前年的天数：

```

1 select datediff(d,curr_year,dateadd(yy,1,curr_year))
2   from (
3 select dateadd(d,-datepart(dy,getdate()+1,getdate()) curr_year
4   from t1
5   ) x

```

## 讨论

### DB2

首先，找到当前年的第一天。使用DAYOFYEAR确定当前日期的年份日期，从当前日期中减去这个值，就得到上一年的最后一天，然后加1即可：

```

select (current_date -
       dayofyear(current_date) day +
       1 day) curr_year
  from t1

CURR_YEAR
-----
01-JAN-2005

```

至此，已经知道了当前年的第一天，只需再加1年，就能得到第二年的第一天。然后从第二年的第一天减去当前年的第一天。

## Oracle

首先找到当前年的第一天。要实现这种功能，只需调用内置 TRUNC 函数，并把它第二个参数设置为 'Y'（从而把日期截断为当前年的第一天）：

```
select select trunc(sysdate,'y') curr_year
      from dual
CURR_YEAR
-----
01-JAN-2005
```

然后加1年，即可得到第二年的第一天。最后，对两个日期进行减法操作，就能得到当前年的天数。

## PostgreSQL

首先，找到当前年的第一天。要实现这种功能，只需调用 DATE\_TRUNC 函数，如下所示：

```
select cast(date_trunc('year',current_date) as date) as curr_year
      from t1
CURR_YEAR
-----
01-JAN-2005
```

然后加1年，就得到了第二年的第一天。最后，对两个日期进行减法操作，而且要确保用较晚日期减去较早日期，就能得到当前年的天数。

## MySQL

首先找到当前年的第一天。使用 DAYOFYEAR 获取当天的年份日期，从当前日期中减去这个值，再加1：

```
select adddate(current_date,-dayofyear(current_date)+1) curr_year
      from t1
CURR_YEAR
-----
01-JAN-2005
```

至此，已经得到了当前年的第一天，下一步就是加1年，以便获得第二年的第一天。然后从第二年的第一天中减去当前年的第一天。其结果就是当前年的天数。

## SQL Server

首先找到当前年的第一天。使用 DATEADD 和 DATEPART，从当前日期中减去当前日期的年份日期，再加1：

```
select dateadd(d,-datepart(dy,getdate()+1,getdate()) curr_year
```

```

from t1
CURR_YEAR
-----
01-JAN-2005

```

至此，已经得到了当前年的第一天，下一步就是加1年，以便获得第二年的第一天。然后从第二年的第一天中减去当前年的第一天。其结果就是当前年的天数。

## 9.3 从日期中提取时间的各部分

### 问题

把当前日期分为6部分：日、月、年、秒、分、时，而且以数字方式返回结果。

### 解决方案

这里使用了当前日期，也可以把该方案应用于其他日期。第1章曾强调了学习及使用RDBMS内置函数的重要性；对日期操作来说更是如此。要从日期中提取时间单位，除了本节讲述的方法之外，还有很多其他方法。试着采用不同的技巧和函数会提高自身水平。

#### DB2

DB2提供了一组内置函数，使用户可以更容易地提取日期的各部分，这些函数分别为HOUR、MINUTE、SECOND、DAY、MONTH和YEAR，依据要返回的时间单位选择函数：如果想要“日”，则使用DAY；想要“时”，则用HOUR等等。例如：

```

1 select  hour( current_timestamp ) hr,
2         minute( current_timestamp ) min,
3         second( current_timestamp ) sec,
4         day( current_timestamp ) dy,
5         month( current_timestamp ) mth,
6         year( current_timestamp ) yr
7   from t1

```

HR	MIN	SEC	DY	MTH	YR
20	28	36	15	6	2005

#### Oracle

使用函数TO\_CHAR和TO\_NUMBER，可以从一个日期中返回指定单位的时间：

```

1 select  to_number(to_char(sysdate,'hh24')) hour,
2         to_number(to_char(sysdate,'mi')) min,
3         to_number(to_char(sysdate,'ss')) sec,
4         to_number(to_char(sysdate,'dd')) day,
5         to_number(to_char(sysdate,'mm')) mth,
6         to_number(to_char(sysdate,'yyyy')) year
7   from dual

```

HOUR	MIN	SEC	DAY	MTH	YEAR
20	28	36	15	6	2005

## PostgreSQL

使用函数 TO\_CHAR 和 TO\_NUMBER, 可以从一个日期中返回指定单位的时间:

```

1 select to_number(to_char(current_timestamp,'hh24'),'99') as hr,
2        to_number(to_char(current_timestamp,'mi'),'99') as min,
3        to_number(to_char(current_timestamp,'ss'),'99') as sec,
4        to_number(to_char(current_timestamp,'dd'),'99') as day,
5        to_number(to_char(current_timestamp,'mm'),'99') as mth,
6        to_number(to_char(current_timestamp,'yyyy'),'9999') as yr
7   from t1

```

HR	MIN	SEC	DAY	MTH	YR
20	28	36	15	6	2005

## MySQL

使用 DATE\_FORMAT 函数, 可以从一个日期中返回指定单位的时间:

```

1 select date_format(current_timestamp,'%k') hr,
2        date_format(current_timestamp,'%i') min,
3        date_format(current_timestamp,'%s') sec,
4        date_format(current_timestamp,'%d') dy,
5        date_format(current_timestamp,'%m') mon,
6        date_format(current_timestamp,'%Y') yr
7   from t1

```

HR	MIN	SEC	DY	MTH	YR
20	28	36	15	6	2005

## SQL Server

使用函数 DATEPART, 可以从一个日期中返回指定单位的时间:

```

1 select datepart( hour, getdate()) hr,
2        datepart( minute,getdate()) min,
3        datepart( second,getdate()) sec,
4        datepart( day, getdate()) dy,
5        datepart( month, getdate()) mon,
6        datepart( year, getdate()) yr
7   from t1

```

HR	MIN	SEC	DY	MTH	YR
20	28	36	15	6	2005

## 讨论

在这些解决方案中, 没有特殊内容, 只是利用以前学过的知识。花些时间学习日期函数可能会有用。本节仅仅粗浅地介绍了每个解决方案中涉及到的函数。实际上, 每个函数都需要很多参数, 而且返回的信息比本节提到的要多。

## 9.4 确定某个月的第一天和最后一天

### 问题

确定当前月的第一天和最后一天。



## 解决方案

解决方案是找到当前月的第一天和最后一天。这里选择了当前月，只需对它进行一些调整，就能够形成对其他月的解决方案。

### DB2

使用 DAY 函数，可以返回当前日期的月份日期，从当前日期中减去这个值，然后加 1，就能得到当前月的第一天。要得到当前月的最后一天，先给当前日期加一个月。然后从这个值中减去由 DAY 函数（针对当前日期）返回的值：

```
1 select (current_date - day(current_date) day +1 day) firstday,
2        (current_date +1 month -day(current_date) day) lastday
3   from t1
```

### Oracle

使用函数 TRUNC，得到当前月的第一天，使用函数 LAST\_DAY，得到当前月的最后一天：

```
1 select trunc(sysdate,'mm') firstday,
2        last_day(sysdate) lastday
3   from dual
```

---

注意：这里的 TRUNC 会丢掉时间部分，而 LAST\_DAY 将保留时间部分。

---

### PostgreSQL

使用 DATE\_TRUNC 函数，可以从当前日期中截取当前月的第一天。一旦获取了当前月的第一天，那么加一个月，并减去 1 天，即可得到当前月的最后一天：

```
1 select firstday,
2        cast(firstday + interval '1 month'
3              - interval '1 day' as date) as lastday
4   from (
5 select cast(date_trunc('month',current_date) as date) as firstday
6   from t1
7  ) x
```

### MySQL

使用 DAY 函数可得到当前日期的月份日期，然后用 DATE\_ADD 从当前日期中减去这个值，并加 1，就能得到当前月的第一天，要得到当前月的最后一天，可使用 LAST\_DAY 函数：

```
1 select date_add(current_date,
2                 interval -day(current_date)+1 day) firstday,
3        last_day(current_date) lastday
4   from t1
```

### SQL Server

使用 DAY 函数，可得到当前日期的月份日期，然后用 DATE\_ADD 从当前日期中减去这

个值，并加1，就能得到当前月的第一天；要得到当前月的最后一天，可给当前日期加一个月，然后从这个值中减去由DAY函数（针对当前日期）返回的值，这里还是使用函数DAY和DATEADD：

```
1 select dateadd(day,-day(getdate()+1,getdate()) firstday,  
2         dateadd(day,  
3                 -day(getdate()),  
4                 dateadd(month,1,getdate())) lastday  
5   from t1
```

## 讨论

### DB2

要得到当前月的第一天，需使用DAY函数。DAY函数能够返回当前日期的月份日期；如果从当前日期中减掉了DAY(CURRENT\_DATE)返回的值，则可得到前一个月的最后一天；再加1天，就能够得到当前月的第一天。要得到当前月的最后一天，需给当前日期加一个月；所得日期在下一个月的月份日期与当前日期在当前月的月份日期一样（如果下一个月比当前月短，数学计算的结果依然正确）。然后，减去由DAY(CURRENT\_DATE)返回的值，便得到当前月的最后一天。

### Oracle

要得到当前月的第一天，可使用TRUNC函数，并把它第二个参数设置为“mm”，以将当前日期“截取”为当前月的第一天；要获得当前月的最后一天，只需使用LAST\_DAY函数。

### PostgreSQL

要得到当前月的第一天，可使用DATE\_TRUNC函数，并把它第二个参数设置为“month”，以将当前日期“截取”为当前月的第一天；要获得当前月的最后一天，给当前月的第一天加1个月，然后减去1天即可。

### MySQL

要得到当前月的第一天，可使用DAY函数。DAY函数能够返回当前日期的月份日期；如果从当前日期中减掉了DAY(CURRENT\_DATE)返回的值，则可得到前一个月的最后一天，再加1天，就能够得到当前月的第一天；要得到当前月的最后一天，只需使用LAST\_DAY函数。

### SQL Server

要得到当前月的第一天，可使用DAY函数。DAY函数能够返回当前日期的月份日期；如果从当前日期中减去DAY(GETDATE())返回的值，则可得到前一个月的最后一天，再加1天，就能够得到当前月的第一天；要得到当前月的最后一天，可使用DATEADD函数，给当前日期加一个月，然后，减去由DAY(GETDATE())返回的值。

## 9.5 确定一年内属于周内某一天的所有日期

### 问题

找出一年内属于周内某一天（星期日或星期一或……或星期六）的所有日期。例如，列出当年中的所有星期五的日期。

### 解决方案

无论哪个数据库系统，解决方案的关键是返回当前年的每一天，而且只保留符合要求的日期。下面的解决方案例子都保留了所有的星期五。

#### DB2

使用递归 WITH 子句，返回当前年的每一天。然后使用函数 DAYNAME 保留星期五：

```
1  with x (dy,yr)
2    as (
3  select dy, year(dy) yr
4    from (
5  select (current_date -
6         dayofyear(current_date) days +1 days) as dy
7    from t1
8         ) tmp1
9    union all
10 select dy+1 days, yr
11    from x
12   where year(dy +1 day) = yr
13   )
14 select dy
15    from x
16   where dayname(dy) = 'Friday'
```

#### Oracle

使用递归 CONNECT BY 子句，返回当前年的每一天。然后使用函数 TO\_CHAR 保留星期五：

```
1  with x
2    as (
3  select trunc(sysdate,'y')+level-1 dy
4    from t1
5   connect by level <=
6             add_months(trunc(sysdate,'y'),12)-trunc(sysdate,'y')
7   )
8  select *
9    from x
10   where to_char( dy, 'dy') = 'fri'
```

#### PostgreSQL

使用函数 GENERATE\_SERIES，返回当前年的每一天。然后使用函数 TO\_CHAR 保留星期五：

```
1 select cast(date_trunc('year',current_date) as date)
2    + x.id as dy
3    from generate_series (
```

```

4      0,
5      ( select cast(
6          cast(
7              date_trunc('year',current_date) as date)
8              + interval '1 years' as date)
9              - cast(
10                 date_trunc('year',current_date) as date) )-1
11      ) x(id)
12 where to_char(
13         cast(
14         date_trunc('year',current_date)
15         as date)+x.id,'dy') = 'fri'

```

## MySQL

使用基干表 T500，返回当前年的每一天。然后使用函数 DAYNAME 保留星期五：

```

1  select dy
2  from (
3  select adddate(x.dy,interval t500.id-1 day) dy
4  from (
5  select dy, year(dy) yr
6  from (
7  select adddate(
8      adddate(current_date,
9      interval -dayofyear(current_date) day),
10     interval 1 day ) dy
11  from t1
12  ) tmp1
13  ) x,
14  t500
15 where year(adddate(x.dy,interval t500.id-1 day)) = x.yr
16  ) tmp2
17 where dayname(dy) = 'Friday'

```

## SQL Server

使用递归 WITH 子句，返回当前年的每一天。然后使用函数 DAYNAME 保留星期五：

```

1  with x (dy,yr)
2  as (
3  select dy, year(dy) yr
4  from (
5  select getdate()-datepart(dy,getdate())+1 dy
6  from t1
7  ) tmp1
8  union all
9  select dateadd(dd,1,dy), yr
10 from x
11 where year(dateadd(dd,1,dy)) = yr
12 )
13 select x.dy
14 from x
15 where datename(dw,x.dy) = 'Friday'
16 option (maxrecursion 400)

```

## 讨论

### DB2

要获得当前年的所有星期五，必须能返回当前年的每一天。首先，使用 DAYOFYEAR 函

数得到当前年的第一天：从当前日期中减去由DAYOFYEAR(CURRENT\_DATE)返回的值，以便得到前一年的12月31日，然后加1，就能得到当前年的第一天：

```
select (current_date -
        dayofyear(current_date) days +1 days) as dy
from t1

DY
-----
01-JAN-2005
```

获取了当前年的第一天之后，使用 WITH 子句，可以给当前年的第一天重复加1天，直到超出当前年为止，其结果集就是当前年的每一天（下面列出了由递归视图 X 返回的部分行）：

```
with x (dy,yr)
as (
select dy, year(dy) yr
from (
select (current_date -
        dayofyear(current_date) days +1 days) as dy
from t1
) tmp1
union all
select dy+1 days, yr
from x
where year(dy +1 day) = yr
)
select dy
from x

DY
-----
01-JAN-2005
...
15-FEB-2005
...
22-NOV-2005
...
31-DEC-2005
```

最后一步，使用 DAYNAME 函数保留属于星期五的那些行。

## Oracle

要获得当前年的所有星期五，必须能返回当前年的每一天。首先，使用 TRUNC 函数得到当前年的第一天：

```
select trunc(sysdate,'y') dy
from t1

DY
-----
01-JAN-2005
```

然后，使用 CONNECT BY 子句返回当前年的每一天（要了解如何使用 CONNECT BY 生成行，请参阅第10章第10.5节）。

注意：另外，尽管本方案使用了 WITH 子句，但也可以使用内联视图。

在编写本书时，Oracle 的 WITH 子句并不能实现递归操作（与 DB2 和 SQL Server 不同），递归操作是用 CONNECT BY 完成的。由视图 X 返回的部分结果集如下所示：

```

with x
as (
select trunc(sysdate,'y')+level-1 dy
from t1
connect by level <=
add_months(trunc(sysdate,'y'),12)-trunc(sysdate,'y')
)
select *
from x

DY
-----
01-JAN-2005
...
15-FEB-2005
...
22-NOV-2005
...
31-DEC-2005

```

最后一步，使用 TO\_CHAR 函数保留星期五。

## PostgreSQL

要获得当前年的所有星期五，必须能返回当前年的每一天。要实现这种功能，可使用 GENERATE\_SERIES 函数。由 GENERATE\_SERIES 函数返回的第一个值为 0，最后一个值为当前年的天数减 1。传递给 GENERATE\_SERIES 函数的第一个参数是 0，而第二个参数是一个查询，该查询确定了当前年的天数（因为是在当前年的第一天上进行加操作，实际上要加的数应该比当前年的天数小 1，这样就不会超出范围进入第二年）。由 GENERATE\_SERIES 函数第二个参数返回的结果如下所示：

```

select cast(
    cast(
        date_trunc('year',current_date) as date)
        + interval '1 years' as date)
    - cast(
        date_trunc('year',current_date) as date)-1 as cnt
from t1

CNT
---
364

```

记住上面的结果集，在 FROM 子句中调用了 GENERATE\_SERIES 函数，如下：GENERATE\_SERIES ( 0, 364 )。如果是闰年，例如 2004，那么第二个参数将为 365。

生成了包含一年日期的列表之后，把由 GENERATE\_SERIES 返回的值与当前年的第一天相加。下面给出了部分结果：

```

select cast(date_trunc('year',current_date) as date)
+ x.id as dy
from generate_series (
    0,
    ( select cast(

```

```

        cast(
            date_trunc('year',current_date) as date)
        + interval '1 years' as date)
    - cast(
        date_trunc('year',current_date) as date) )-1
    ) x(id)

DY
-----
01-JAN-2005
...
15-FEB-2005
...
22-NOV-2005
...
31-DEC-2005

```

最后一步，使用 TO\_CHAR 函数保留星期五。

## MySQL

要获得当前年的所有星期五，必须能返回当前年的每一天。首先，使用 DAYOFYEAR 函数得到当前年的第一天：然后从当前日期中减去由 DAYOFYEAR(CURRENT\_DATE) 返回的值，再加 1，就能得到当前年的第一天：

```

select adddate(
    adddate(current_date,
        interval -dayofyear(current_date) day),
        interval 1 day ) dy
    from t1

DY
-----
01-JAN-2005

```

然后，使用表 T500 生成足够多的行，以便返回当前年的每一天。要实现这种功能，需把 T500.ID 的每个值都与当前年的第一天相加，直到超出当前年为止。下面给出了这种操作的部分结果：

```

select adddate(x.dy,interval t500.id-1 day) dy
    from (
select dy, year(dy) yr
    from (
select adddate(
    adddate(current_date,
        interval -dayofyear(current_date) day),
        interval 1 day ) dy
    from t1
    ) tmp1
    ) x,
    t500
where year(adddate(x.dy,interval t500.id-1 day)) = x.yr

DY
-----
01-JAN-2005
...
15-FEB-2005
...
22-NOV-2005
...
31-DEC-2005

```

最后一步，使用 DAYNAME 函数保留星期五。

## SQL Server

要获得当前年的所有星期五，必须能返回当前年的每一天。首先，使用 DATEPART 函数得到当前年的第一天：然后从当前日期中减去由 DATEPART(DY, GETDATE()) 返回的值，再加 1，就能得到当前年的第一天：

```
select getdate()-datepart(dy,getdate())+1 dy
from t1

DY
-----
01-JAN-2005
```

获取了当前年的第一天之后，使用 WITH 子句和 DATEADD 函数，给当前年的第一天重复加 1 天，直到超出当前年为止，其结果集就是当前年的每一天（下面列出了由递归视图 X 返回的部分行）：

```
with x (dy,yr)
as (
select dy, year(dy) yr
from (
select getdate()-datepart(dy,getdate())+1 dy
from t1
) tmp1
union all
select dateadd(dd,1,dy), yr
from x
where year(dateadd(dd,1,dy)) = yr
)
select x.dy
from x
option (maxrecursion 400)

DY
-----
01-JAN-2005
...
15-FEB-2005
...
22-NOV-2005
...
31-DEC-2005
```

最后，使用 DATENAME 函数保留属于星期五的那些行。要保证这种解决方案正确运行，MAXRECURSION 至少要设置为 366（递归视图 X 中针对当前年的年份的筛选条件，确保生成的行不会超过 366）。

## 9.6 确定某月内第一个和最后一个“周内某天”的日期

### 问题

例如，找出当前月的第一个星期一及最后一个星期一的日期。



## 解决方案

这里选用了Monday和当前月,也可以将该解决方案应用于其他日子和月份。由于每个相同的周内日期的间隔都是7天,所以知道第一个后,加7天就能得到第二个,加14天就能得到第三个。同样,如果知道某个月的最后一个指定的周内日期,则减7就能得到第三个,再减7就能得到第二个。

## DB2

使用递归WITH子句,可生成当前月的每一天,用CASE表达式,可标记所有星期一。第一个及最后一个星期一分别就是第一个及最后一个标记的日期:

```

1  with x (dy,mth,is_monday)
2  as (
3  select dy,month(dy),
4         case when dayname(dy)='Monday'
5              then 1 else 0
6         end
7         from (
8  select (current_date-day(current_date) day +1 day) dy
9         from t1
10        ) tmp1
11  union all
12  select (dy +1 day), mth,
13         case when dayname(dy +1 day)='Monday'
14              then 1 else 0
15         end
16         from x
17         where month(dy +1 day) = mth
18  )
19  select min(dy) first_monday, max(dy) last_monday
20         from x
21  where is_monday = 1

```

## Oracle

使用函数NEXT\_DAY和LAST\_DAY,加上聪明的日期算法,就能够得到当前月的第一个星期一及最后一个星期一:

```

select next_day(trunc(sysdate,'mm')-1,'MONDAY') first_monday,
       next_day(last_day(trunc(sysdate,'mm'))-7,'MONDAY') last_monday
from dual

```

## PostgreSQL

使用函数DATE\_TRUNC,找到当前月的第一天。得到当前月的第一天之后,可以使用简单的算法和表示星期几的数字值(星期日~星期六分别对应1~7),以获得当前月的第一个和最后一个星期一:

```

1  select first_monday,
2         case to_char(first_monday+28,'mm')
3             when mth then first_monday+28
4                  else first_monday+21
5         end as last_monday
6         from (
7  select case sign(cast(to_char(dy,'d') as integer)-2)
8         when 0

```

```

9          then dy
10         when -1
11         then dy+abs(cast(to_char(dy,'d') as integer)-2)
12         when 1
13         then (7-(cast(to_char(dy,'d') as integer)-2))+dy
14     end as first_monday,
15     mth
16   from (
17   select cast(date_trunc('month',current_date) as date) as dy,
18          to_char(current_date,'mm') as mth
19   from t1
20        ) x
21        ) y

```

## MySQL

使用函数 `ADDDATE`，找到当前月的第一天。得到当前月的第一天之后，可以使用简单的算法和表示星期几的数字值（星期日~星期六分别对应1~7），以获得当前月的第一个星期一和最后一个星期一：

```

1  select first_monday,
2         case month(adddate(first_monday,28))
3             when mth then adddate(first_monday,28)
4             else adddate(first_monday,21)
5         end last_monday
6   from (
7   select case sign(dayofweek(dy)-2)
8          when 0 then dy
9          when -1 then adddate(dy,abs(dayofweek(dy)-2))
10         when 1 then adddate(dy,(7-(dayofweek(dy)-2)))
11        end first_monday,
12        mth
13   from (
14   select adddate(adddate(current_date,-day(current_date)),1) dy,
15          month(current_date) mth
16   from t1
17        ) x
18        ) y

```

## SQL Server

使用递归 `WITH` 子句，生成当前月的每一天，然后，使用 `CASE` 表达式，标记所有星期一。第一个及最后一个星期一分别就是已标记日期中的第一个和最后一个：

```

1  with x (dy,mth,is_monday)
2  as (
3  select dy,mth,
4         case when datepart(dw,dy) = 2
5             then 1 else 0
6         end
7   from (
8   select dateadd(day,1,dateadd(day,-day(getdate()),getdate())) dy,
9          month(getdate()) mth
10  from t1
11       ) tmp1
12  union all
13  select dateadd(day,1,dy),
14         mth,
15         case when datepart(dw,dateadd(day,1,dy)) = 2
16             then 1 else 0
17         end
18  from x
19  where month(dateadd(day,1,dy)) = mth

```

```

20 )
21 select min(dy) first_monday,
22        max(dy) last_monday
23   from x
24  where is_monday = 1

```

## 讨论

### DB2 和 SQL Server

尽管 DB2 和 SQL Server 使用了不同的函数解决这个问题，但所采用的技巧却完全相同。如果仔细观察这两种解决方案，可以看到，二者之间的唯一差别是日期加法的方式不同。下面的讨论涵盖这两种解决方案，而使用 DB2 解决方案的代码显示中间步骤的结果。

---

注意：对不能使用递归 WITH 子句的 SQL Server 或 DB2 版本，也可以采用 Post-greSQL 技巧。

---

要找到当前月的第一个和最后一个星期一，首先应该返回当前月的第一天。递归视图 X 中的内联视图 TMP1 能够返回当前月的第一天，它先获得当前日期，特别是当前日期的月份日期。当前日期的月份日期表示了当前日期是当前月份的第几天（例如，4 月 10 日是 4 月的第 10 天）。如果从当前日期中减去这个值，则会得到前一个月的最后一天（例如，从 4 月 10 日中减去 10，即得到 3 月的最后一天）。进行减法操作之后，再加 1 天，就能得到当前月的第一天：

```

select (current_date-day(current_date) day +1 day) dy
   from t1

```

```

DY
-----
01-JUN-2005

```

然后，使用 MONTH 函数找到当前日期的月份，再使用 CASE 表达式确定当前月的第一天是否为星期一：

```

select dy, month(dy) mth,
       case when dayname(dy)='Monday'
            then 1 else 0
       end is_monday
   from (
select (current_date-day(current_date) day +1 day) dy
   from t1
       ) tmp1

```

```

DY          MTH  IS_MONDAY
-----
01-JUN-2005    6          0

```

接下来，使用 WITH 子句的递归功能，对当前月的第一天进行重复加 1 天操作，直到超出当前月为止，同时用 CASE 表达式确定当前月其中的哪些天是星期一（把星期一标记为“1”）。下面列出了递归视图 X 的部分输出：

```

with x (dy,mth,is_monday)
as (
select dy,month(dy) mth,

```

```

        case when dayname(dy)='Monday'
            then 1 else 0
        end is_monday
    from (
        select (current_date-day(current_date) day +1 day) dy
        from t1
        ) tmp1
    union all
    select (dy +1 day), mth,
        case when dayname(dy +1 day)='Monday'
            then 1 else 0
        end
    from x
    where month(dy +1 day) = mth
)
select *
from x

```

DY	MTH	IS_MONDAY
01-JUN-2005	6	0
02-JUN-2005	6	0
03-JUN-2005	6	0
04-JUN-2005	6	0
05-JUN-2005	6	0
06-JUN-2005	6	1
07-JUN-2005	6	0
08-JUN-2005	6	0
...		

对于 IS\_MONDAY，只有星期一的值为 1。最后一步是针对 IS\_MONDAY 为 1 的行使用聚集函数 MIN 和 MAX，找到当前月的第一个和最后一个星期一。

## Oracle

函数 NEXT\_DAY 使这个问题相当容易解决。要找到当前月第一个星期一，首先调用 TRUNC 函数，并采用某种日期算法，返回前一个月的最后一天：

```

select trunc(sysdate,'mm')-1 dy
from dual

```

DY
31-MAY-2005

然后使用 NEXT\_DAY 函数，找到前一个月最后一天之后的第一个星期一（即当前月的第一个星期一）：

```

select next_day(trunc(sysdate,'mm')-1,'MONDAY') first_monday
from dual

```

FIRST_MONDAY
06-JUN-2005

要找到当前月的最后一个星期一，首先调用 TRUNC 函数返回当前月的第一天：

```

select trunc(sysdate,'mm') dy
from dual

```

DY
01-JUN-2005

下一步，查找当前月的最后一个星期（最后 7 天）。使用 LAST\_DAY 函数，可得到当前月的最后一天，然后减去 7 天：

```
select last_day(trunc(sysdate,'mm'))-7 dy
from dual

DY
-----
23-JUN-2005
```

如果还不很清楚，则可以由当前月的最后一天向前倒推 7 天，在此日期之后，正好各有一个星期日~星期六属于该月份。最后一步就是用函数 NEXT\_DAY 查找下一个（也就是当前月的最后一个）星期一：

```
select next_day(last_day(trunc(sysdate,'mm'))-7,'MONDAY') last_monday
from dual

LAST_MONDAY
-----
27-JUN-2005
```

## PostgreSQL 和 MySQL

PostgreSQL 和 MySQL 采用的解决方案相同，其差别是所调用的函数不同。无论代码多长，它们各自的查询都非常简单，要获得当前月的第一个和最后一个星期一，不需要什么高深的东西。

首先，获取当前月的第一天，然后查找当前月的第一个星期一。由于没有针对给定的周内日期查找下一个日期的函数，因此需要采用一些算法。始于第 7 行的 CASE 表达式（两种解决方案中一样）计算了表示当前月第一天是星期几的数值与表示星期一的数值之差。当采用 'D' 或 'd' 格式时，函数 TO\_CHAR (PostgreSQL) 和函数 DAYOFWEEK (MySQL) 都会返回从 1 到 7 的数字值，正好与星期日到星期六相对应，总是用 2 表示星期一。CASE 先判断上述差的正负号（用 SIGN 函数）。如果 SIGN 返回值符号为 0，那么当前月第一天即为星期一，而且是当前月的第一个星期一；如果计算结果为 -1，那么当前月第一天为星期日，只需对它加上 2 和 1 的差（分别表示星期一和星期日），就能得到当前月的第一个星期一；

---

**注意：**如果不能理解这些内容，那么忘记星期几，只进行算术运算。例如，起始日期为星期二，想要查找下一个星期五。当使用了带有 'd' 格式的函数 TO\_CHAR 或函数 DAYOFWEEK 时，星期五的数字值为 6，星期二的数字值为 3。从 6 到 3，只需进行减法操作（6-3=3），并与最小值相加（(6-3)+3=6）。这样，不管实际日期是什么，如果起始日期的数字值比要查找日期的数字值小，则将两个日期之差加上当前日期的数字值，就能得到要查找的日期。

---

如果 SIGN 结果为 1，那么当前月的第一天在星期二和星期六之间（包括这两天）。因为表示当前月第一天为星期几的数值大于 2（星期一），故用 7 减去该数值与表示星期一的数值（2），将其差与当前月第一天相加这就是要查找的那个周内日期，本例中是星期一。

注意：如果不能理解这些内容，那么忘记星期几，只进行算术运算。例如，起始日期为星期五，想要查找下一个星期二。星期二的数字值（3）比星期五的数字值（6）要小，要从6到3，需要从7中减去两个数字值之差（ $7 - (13-6) = 4$ ），并把结果（4）与起始日星期五相加。（在13-6中的竖线用于返回差的绝对值）。这里，并没有把4和6相加（等于10），而是给星期五加了4天，最后结果为下个星期二。

藏在CASE表达式后面的是给PostgreSQL和MySQL创建某种“下一天”函数的思想。如果起始日不是当前月第一天，DY值是由CURRENT\_DATE返回的值，而且CASE表达式的结果为相对于当前日期的下个星期一（如果CURRENT\_DATE是星期一，则会返回该日期）。

得到了当前月的第一个星期一之后，加21天或28天，就能找到当前月的最后一个星期一。第2~5行的CASE表达式用于检查加28之后是否超出了本月范围，从而确定是加21天还是28天。CASE表达式采用如下步骤实现此功能：

1. 它给FIRST\_MONDAY值加28。
2. 使用TO\_CHAR (PostgreSQL)或MONTH函数，CASE表达式将从FIRST\_MONDAY + 28结果中提取当前月名。
3. 把第二步产生的结果与内联视图的MTH值相比较。MTH值是当前月份名，它来自CURRENT\_DATE。如果两个值匹配，则加28天不会超出当前月，CASE表达式返回FIRST\_MONDAY + 28；如果两个值不匹配，则不能加28天，CASE表达式返回FIRST\_MONDAY + 21。这种计算非常方便，只有21和28两种选择。

注意：也可以扩展该解决方案，即加7和14天就能分别找到当前月的第二个和第三个星期一。

## 9.7 创建日历

### 问题

为当前月创建一个日历。日历的格式应该与书桌上的日历相似，通常情况下包含7列、5行。

### 解决方案

每个解决方案看起来都有所不同，但它们都采用了同样方式解决问题：返回当前月的每一天，然后依据当前月中每周的周内日期创建日历。

日历可采用不同的格式。例如Unix的cal命令采用从星期日到星期六的格式。本节中的例子都以ISO周序号为基准，因此采用了星期一到星期五的格式最方便。一旦习惯了这

种解决方案, 就会明白, 无论怎么重新定义格式都是非常简单的事, 只需修改由 ISO 周规定的值即可。

注意: 刚开始用 SQL 创建不同格式的可读输出时, 会发现查询变得很长, 不要被这些查询吓倒, 如果将本节给出的查询分解开来分段运行, 它们都非常简单。

## DB2

使用递归 WITH 子句, 返回当前月的每一天。然后使用 CASE 和 MAX 转换为周内日期:

```

1  with x(dy, dm, mth, dw, wk)
2  as (
3  select (current_date - day(current_date) day + 1 day) dy,
4         day((current_date - day(current_date) day + 1 day)) dm,
5         month(current_date) mth,
6         dayofweek(current_date - day(current_date) day + 1 day) dw,
7         week_iso(current_date - day(current_date) day + 1 day) wk
8  from t1
9  union all
10 select dy+1 day, day(dy+1 day), mth,
11        dayofweek(dy+1 day), week_iso(dy+1 day)
12  from x
13 where month(dy+1 day) = mth
14 )
15 select max(case dw when 2 then dm end) as Mo,
16        max(case dw when 3 then dm end) as Tu,
17        max(case dw when 4 then dm end) as We,
18        max(case dw when 5 then dm end) as Th,
19        max(case dw when 6 then dm end) as Fr,
20        max(case dw when 7 then dm end) as Sa,
21        max(case dw when 1 then dm end) as Su
22  from x
23 group by wk
24 order by wk

```

## Oracle

使用递归的 CONNECT BY 子句, 返回当前月的每一天。然后使用 CASE 和 MAX 转换为周内日期:

```

1  with x
2  as (
3  select *
4  from (
5  select to_char(trunc(sysdate, 'mm')+level-1, 'iw') wk,
6         to_char(trunc(sysdate, 'mm')+level-1, 'dd') dm,
7         to_number(to_char(trunc(sysdate, 'mm')+level-1, 'd')) dw,
8         to_char(trunc(sysdate, 'mm')+level-1, 'mm') curr_mth,
9         to_char(sysdate, 'mm') mth
10  from dual
11  connect by level <= 31
12  )
13  where curr_mth = mth
14  )
15 select max(case dw when 2 then dm end) Mo,
16        max(case dw when 3 then dm end) Tu,
17        max(case dw when 4 then dm end) We,
18        max(case dw when 5 then dm end) Th,
19        max(case dw when 6 then dm end) Fr,
20        max(case dw when 7 then dm end) Sa,

```

```

21      max(case dw when 1 then dm end) Su
22  from x
23  group by wk
24  order by wk

```

## PostgreSQL

使用函数 `GENERATE_SERIES` 返回当前月的每一天。然后使用 `MAX` 和 `CASE` 转换为周内日期：

```

1  select max(case dw when 2 then dm end) as Mo,
2         max(case dw when 3 then dm end) as Tu,
3         max(case dw when 4 then dm end) as We,
4         max(case dw when 5 then dm end) as Th,
5         max(case dw when 6 then dm end) as Fr,
6         max(case dw when 7 then dm end) as Sa,
7         max(case dw when 1 then dm end) as Su
8  from (
9  select *
10 from (
11 select cast(date_trunc('month',current_date) as date)+x.id,
12        to_char(
13          cast(
14            date_trunc('month',current_date)
15              as date)+x.id,'iw') as wk,
16        to_char(
17          cast(
18            date_trunc('month',current_date)
19              as date)+x.id,'dd') as dm,
20        cast(
21          to_char(
22            cast(
23              date_trunc('month',current_date)
24                as date)+x.id,'d') as integer) as dw,
25        to_char(
26          cast(
27            date_trunc('month',current_date)
28              as date)+x.id,'mm') as curr_mth,
29        to_char(current_date,'mm') as mth
30      from generate_series (0,31) x(id)
31     ) x
32  where mth = curr_mth
33     ) y
34  group by wk
35  order by wk

```

## MySQL

使用表 `T500` 返回当前月的每一天。然后使用 `MAX` 和 `CASE` 转换为周内日期：

```

1  select max(case dw when 2 then dm end) as Mo,
2         max(case dw when 3 then dm end) as Tu,
3         max(case dw when 4 then dm end) as We,
4         max(case dw when 5 then dm end) as Th,
5         max(case dw when 6 then dm end) as Fr,
6         max(case dw when 7 then dm end) as Sa,
7         max(case dw when 1 then dm end) as Su
8  from (
9  select date_format(dy,'%u') wk,
10         date_format(dy,'%d') dm,
11         date_format(dy,'%w')+1 dw
12  from (
13  select adddate(x.dy,t500.id-1) dy,
14         x.mth

```



```
15 from (
16 select adddate(current_date,-dayofmonth(current_date)+1) dy,
17        date_format(
18            adddate(current_date,
19                -dayofmonth(current_date)+1),
20                '%m') mth
21 from t1
22 ) x,
23 t500
24 where t500.id <= 31
25 and date_format(adddate(x.dy,t500.id-1),'%m') = x.mth
26 ) y
27 ) z
28 group by wk
29 order by wk
```

## SQL Server

使用递归 WITH 子句，返回当前月的每一天。然后使用 CASE 和 MAX 转换为周内日期：

```
1 with x(dy,dm,mth,dw,wk)
2 as (
3 select dy,
4        day(dy) dm,
5        datepart(m,dy) mth,
6        datepart(dw,dy) dw,
7        case when datepart(dw,dy) = 1
8            then datepart(dw,dy)-1
9            else datepart(dw,dy)
10       end wk
11 from (
12 select dateadd(day,-day(getdate()+1,getdate()) dy
13 from t1
14 ) x
15 union all
16 select dateadd(d,1,dy), day(dateadd(d,1,dy)), mth,
17        datepart(dw,dateadd(d,1,dy)),
18        case when datepart(dw,dateadd(d,1,dy)) = 1
19            then datepart(dw,dateadd(d,1,dy))-1
20            else datepart(dw,dateadd(d,1,dy))
21       end
22 from x
23 where datepart(m,dateadd(d,1,dy)) = mth
24 )
25 select max(case dw when 2 then dm end) as Mo,
26        max(case dw when 3 then dm end) as Tu,
27        max(case dw when 4 then dm end) as We,
28        max(case dw when 5 then dm end) as Th,
29        max(case dw when 6 then dm end) as Fr,
30        max(case dw when 7 then dm end) as Sa,
31        max(case dw when 1 then dm end) as Su
32 from x
33 group by wk
34 order by wk
```

## 讨论

### DB2

首先，对于要创建日历的月份，返回它的每一天。使用递归 WITH 子句（如果无法使用 WITH，则可以使用基干表，例如 MySQL 解决方案中的 T500）可实现该功能。对于当前月的每一天（别名 DM），需要返回日期的不同部分：星期几（别名 DW）、当前月份（别

名 MTH)、ISO 周 (别名 WK)。递归视图 X 在递归之前 (UNION ALL 的上半部分) 的结果如下所示:

```
select (current_date -day(current_date) day +1 day) dy,
       day((current_date -day(current_date) day +1 day)) dm,
       month(current_date) mth,
       dayofweek(current_date -day(current_date) day +1 day) dw,
       week_iso(current_date -day(current_date) day +1 day) wk
from t1
```

DY	DM	MTH	DW	WK
01-JUN-2005	01	06	4	22

下一步重复递增 DM 值 (递增次数就是月份天数), 直到超出当前月为止。在当前月的每一天进行处理的同时, 也得到当天对应星期几、它属于哪个 ISO 周, 下面给出了部分结果:

```
with x(dy, dm, mth, dw, wk)
as (
select (current_date -day(current_date) day +1 day) dy,
       day((current_date -day(current_date) day +1 day)) dm,
       month(current_date) mth,
       dayofweek(current_date -day(current_date) day +1 day) dw,
       week_iso(current_date -day(current_date) day +1 day) wk
from t1
union all
select dy+1 day, day(dy+1 day), mth,
       dayofweek(dy+1 day), week_iso(dy+1 day)
from x
where month(dy+1 day) = mth
)
select *
from x
```

DY	DM	MTH	DW	WK
01-JUN-2005	01	06	4	22
02-JUN-2005	02	06	5	22
...				
21-JUN-2005	21	06	3	25
22-JUN-2005	22	06	4	25
...				
30-JUN-2005	30	06	5	26

此时, 当前月的每一天包含下列信息: 当月每天的日期、两位数字的月份日期值、两位数字的月份值、一位数字表示的星期几 (1~7 分别对应星期日~星期六) 以及两位数字表示的 ISO 周次。有了这些信息, 就可以使用 CASE 表达式确定 DM (当前月的每一天) 中每个值对应星期几。下面给出了部分结果:

```
with x(dy, dm, mth, dw, wk)
as (
select (current_date -day(current_date) day +1 day) dy,
       day((current_date -day(current_date) day +1 day)) dm,
       month(current_date) mth,
       dayofweek(current_date -day(current_date) day +1 day) dw,
       week_iso(current_date -day(current_date) day +1 day) wk
from t1
union all
select dy+1 day, day(dy+1 day), mth,
       dayofweek(dy+1 day), week_iso(dy+1 day)
from x
where month(dy+1 day) = mth
)
```

```

        from x
        where month(dy+1 day) = mth
    )
    select wk,
        case dw when 2 then dm end as Mo,
        case dw when 3 then dm end as Tu,
        case dw when 4 then dm end as We,
        case dw when 5 then dm end as Th,
        case dw when 6 then dm end as Fr,
        case dw when 7 then dm end as Sa,
        case dw when 1 then dm end as Su
    from x

```

```

WK MO TU WE TH FR SA SU
-- -- -- -- -- -- --
22      01
22      02
22      03
22      04
22      05
23 06
23      07
23      08
23      09
23      10
23      11
23      12

```

从上面的输出可以看出，每周的每一天都分别返回一行。现在需要做的是按照周次给这些日期分组，然后把每周的所有日期都放入一行中。使用聚集函数MAX，并按照WK(ISO周)分组，就可以把一周的所有日期作为一行返回。对结果按WK排序，以保证日历的格式和所有日期的顺序正确。最终输出结果如下：

```

with x(dy,dm,mth,dw,wk)
as (
select (current_date -day(current_date) day +1 day) dy,
       day((current_date -day(current_date) day +1 day)) dm,
       month(current_date) mth,
       dayofweek(current_date -day(current_date) day +1 day) dw,
       week_iso(current_date -day(current_date) day +1 day) wk
from t1
union all
select dy+1 day, day(dy+1 day), mth,
       dayofweek(dy+1 day), week_iso(dy+1 day)
from x
where month(dy+1 day) = mth
)
select max(case dw when 2 then dm end) as Mo,
       max(case dw when 3 then dm end) as Tu,
       max(case dw when 4 then dm end) as We,
       max(case dw when 5 then dm end) as Th,
       max(case dw when 6 then dm end) as Fr,
       max(case dw when 7 then dm end) as Sa,
       max(case dw when 1 then dm end) as Su
from x
group by wk
order by wk

```

```

MO TU WE TH FR SA SU
-- -- -- -- -- -- --
      01 02 03 04 05
06 07 08 09 10 11 12
13 14 15 16 17 18 19
20 21 22 23 24 25 26
27 28 29 30

```

## Oracle

首先，使用递归 CONNECT BY 子句，对要创建日历的月份的每一天生成一行信息。如果并未使用 Oracle9i Database 或更高版本，则不能按这种方式使用 CONNECT BY，然而，可以使用基干表，例如 MySQL 解决方案中的 T500。

除当前月每天的日期外，还需要每天的其他信息：月份日期（别名 DM）、星期几（别名 DW）、当前月份（别名 MTH）、ISO 周（别名 WK）。WITH 视图 X 为当前月的第一天生成如下结果：

```
select trunc(sysdate,'mm') dy,
       to_char(trunc(sysdate,'mm'),'dd') dm,
       to_char(sysdate,'mm') mth,
       to_number(to_char(trunc(sysdate,'mm'),'d')) dw,
       to_char(trunc(sysdate,'mm'),'iw') wk
from dual
```

DY	DM	MT	DW	WK
01-JUN-2005	01	06	4	22

下一步，重复递增 DM 值（递增次数就是月份天数），直到超出当前月为止。在对当前月的每一天进行处理的同时，也得到每天对应星期几、它属于哪个 ISO 周等信息，下面给出了部分结果：

```
with x
as (
select *
from (
select trunc(sysdate,'mm')+level-1 dy,
       to_char(trunc(sysdate,'mm')+level-1,'iw') wk,
       to_char(trunc(sysdate,'mm')+level-1,'dd') dm,
       to_number(to_char(trunc(sysdate,'mm')+level-1,'d')) dw,
       to_char(trunc(sysdate,'mm')+level-1,'mm') curr_mth,
       to_char(sysdate,'mm') mth
from dual
connect by level <= 31
)
where curr_mth = mth
)
```

```
select *
from x
```

DY	WK	DM	DW	CU	MT
01-JUN-2005	22	01	4	06	06
02-JUN-2005	22	02	5	06	06
...					
21-JUN-2005	25	21	3	06	06
22-JUN-2005	25	22	4	06	06
...					
30-JUN-2005	26	30	5	06	06

此时，为当前月的每一天生成了一行信息，其中包含：两位数字的月份日期、两位数字的月份值、一位数字表示的星期几（1~7 分别对应星期日~星期六）以及两位数字的 ISO 周次。有了这些信息，就可以用 CASE 表达式确定 DM（当前月的每一天）中每个值对应星期几。下面给出了部分结果：

```

with x
as (
select *
from (
select trunc(sysdate,'mm')+level-1 dy,
       to_char(trunc(sysdate,'mm')+level-1,'iw') wk,
       to_char(trunc(sysdate,'mm')+level-1,'dd') dm,
       to_number(to_char(trunc(sysdate,'mm')+level-1,'d')) dw,
       to_char(trunc(sysdate,'mm')+level-1,'mm') curr_mth,
       to_char(sysdate,'mm') mth
  from dual
 connect by level <= 31
)
where curr_mth = mth
)
select wk,
       case dw when 2 then dm end as Mo,
       case dw when 3 then dm end as Tu,
       case dw when 4 then dm end as We,
       case dw when 5 then dm end as Th,
       case dw when 6 then dm end as Fr,
       case dw when 7 then dm end as Sa,
       case dw when 1 then dm end as Su
  from x

```

WK	MO	TU	WE	TH	FR	SA	SU
22			01				
22				02			
22					03		
22						04	
22							05
23	06						
23		07					
23			08				
23				09			
23					10		
23						11	
23							12

从上面的输出可以看出，每周的每一天都分别返回一行信息，而且日期所在位置与这天是星期几相对应。现在需要做的是把每周的所有日期都放入一行中。使用聚集函数 MAX，并按照 WK（ISO 周）分组，就可以把一周的所有日期作为一行返回。对结果按 WK 排序，以保证日历的格式和所有日期的顺序正确。最终输出结果如下：

```

with x
as (
select *
from (
select to_char(trunc(sysdate,'mm')+level-1,'iw') wk,
       to_char(trunc(sysdate,'mm')+level-1,'dd') dm,
       to_number(to_char(trunc(sysdate,'mm')+level-1,'d')) dw,
       to_char(trunc(sysdate,'mm')+level-1,'mm') curr_mth,
       to_char(sysdate,'mm') mth
  from dual
 connect by level <= 31
)
where curr_mth = mth
)
select max(case dw when 2 then dm end) Mo,
       max(case dw when 3 then dm end) Tu,
       max(case dw when 4 then dm end) We,
       max(case dw when 5 then dm end) Th,
       max(case dw when 6 then dm end) Fr,
       max(case dw when 7 then dm end) Sa,

```

```

        max(case dw when 1 then dm end) Su
    from x
    group by wk
    order by wk
MO TU WE TH FR SA SU
-- -- -- -- -- -- --
      01 02 03 04 05
06 07 08 09 10 11 12
13 14 15 16 17 18 19
20 21 22 23 24 25 26
27 28 29 30

```

## PostgreSQL

使用 GENERATE\_SERIES 函数，为当前月的每一天返回一行。如果 PostgreSQL 版本不支持 GENERATE\_SERIES，那么可以按照 MySQL 解决方案中所述的查询基于表方式。

当前月的每一天将返回下列信息：月份日期（别名 DM）、星期几（别名 DW）、当前月份（别名 MTH）、ISO 周次（别名 WK）。由于需要设置格式，还要做显式类型转换，这种解决方案看起来很复杂，但它其实很简单。下面列出了内联视图 X 的部分结果：

```

select cast(date_trunc('month',current_date) as date)+x.id as dy,
       to_char(
         cast(
           date_trunc('month',current_date)
             as date)+x.id,'iw') as wk,
       to_char(
         cast(
           date_trunc('month',current_date)
             as date)+x.id,'dd') as dm,
       cast(
         to_char(
           cast(
             date_trunc('month',current_date)
               as date)+x.id,'d') as integer) as dw,
       to_char(
         cast(
           date_trunc('month',current_date)
             as date)+x.id,'mm') as curr_mth,
       to_char(current_date,'mm') as mth
from generate_series (0,31) x(id)

```

DY	WK	DM	DW	CU	MT
01-JUN-2005	22	01	4	06	06
02-JUN-2005	22	02	5	06	06
...					
21-JUN-2005	25	21	3	06	06
22-JUN-2005	25	22	4	06	06
...					
30-JUN-2005	26	30	5	06	06

应该注意，当处理当前月的每一天时，也返回了它是星期几及 ISO 周序号。要确保只返回指定月份所包含的日期，用 CURR\_MTH = MTH（每一天所属的月份应该是当前日期所在的月份）作为筛选条件。此时，为当前月的每一天的信息包含：两位数字的月份日期、两位数字的月份、一位数字表示星期几（1~7 分别对应星期日~星期六）以及每天所处的 ISO 周序号。接下来，可以使用 CASE 表达式确定 DM（当前月的每一天）中每个值对应星期几。下面给出了部分结果：

```

select case dw when 2 then dm end as Mo,
       case dw when 3 then dm end as Tu,
       case dw when 4 then dm end as We,
       case dw when 5 then dm end as Th,
       case dw when 6 then dm end as Fr,
       case dw when 7 then dm end as Sa,
       case dw when 1 then dm end as Su
from (
select *
from (
select cast(date_trunc('month',current_date) as date)+x.id,
       to_char(
         cast(
           date_trunc('month',current_date)
             as date)+x.id,'iw') as wk,
       to_char(
         cast(
           date_trunc('month',current_date)
             as date)+x.id,'dd') as dm,
       cast(
         to_char(
           cast(
             date_trunc('month',current_date)
               as date)+x.id,'d') as integer) as dw,
       to_char(
         cast(
           date_trunc('month',current_date)
             as date)+x.id,'mm') as curr_mth,
       to_char(current_date,'mm') as mth
from generate_series (0,31) x(id)
) x
where mth = curr_mth
) y

```

	WK	MO	TU	WE	TH	FR	SA	SU
--	--	--	--	--	--	--	--	--
22				01				
22					02			
22						03		
22							04	
22								05
23	06							
23		07						
23			08					
23				09				
23					10			
23						11		
23							12	

从上面的输出结果可以看出，每周的每一天都返回一行，而且日期所在位置与这天是星期几相对应。现在需要做的是把每周的所有日期都放入一行中。然后，使用聚集函数 MAX，并按照 WK（ISO 周序号）进行分组。其结果正如日历所显示的，它把一周的所有日期作为一行返回。对结果按 WK 排序，以保证所有日期的顺序正确。最终输出结果如下：

```

select max(case dw when 2 then dm end) as Mo,
       max(case dw when 3 then dm end) as Tu,
       max(case dw when 4 then dm end) as We,
       max(case dw when 5 then dm end) as Th,
       max(case dw when 6 then dm end) as Fr,
       max(case dw when 7 then dm end) as Sa,
       max(case dw when 1 then dm end) as Su
from (
select *

```

```

from (
select cast(date_trunc('month',current_date) as date)+x.id,
       to_char(
         cast(
           date_trunc('month',current_date)
             as date)+x.id,'iw') as wk,
       to_char(
         cast(
           date_trunc('month',current_date)
             as date)+x.id,'dd') as dm,
       cast(
         to_char(
           cast(
             date_trunc('month',current_date)
               as date)+x.id,'d') as integer) as dw,
       to_char(
         cast(
           date_trunc('month',current_date)
             as date)+x.id,'mm') as curr_mth,
       to_char(current_date,'mm') as mth
  from generate_series (0,31) x(id)
 ) x
where mth = curr_mth
 ) y
group by wk
order by wk

```

```

MO TU WE TH FR SA SU
-- -- -- -- -- -- --
          01 02 03 04 05
06 07 08 09 10 11 12
13 14 15 16 17 18 19
20 21 22 23 24 25 26
27 28 29 30

```

## MySQL

首先，对于要创建日历的月份，返回它的每一天为一行，要查询表 T500。把由 T500 返回的每个值都与当前月的第一天相加，得到当前月的每一天。

对于每个日期，需要返回下列信息位：月份日期（别名 DM）、星期几（别名 DW）、当前月份（别名 MTH）、ISO 周序号（别名 WK）。内联视图 X 返回当前月的第一天以及两位数字的当前月份值，其结果如下所示：

```

select adddate(current_date,-dayofmonth(current_date)+1) dy,
       date_format(
         adddate(current_date,
                  -dayofmonth(current_date)+1),
           '%m') mth
  from t1

```

```

DY          MT
-----
01-JUN-2005 06

```

下一步，从当前月的第一天开始，对当前月的每一天进行处理。应该注意，在处理每一天时，也得到星期几及 ISO 周序号。要确保只返回指定月份所包含的日期，故用 CURR\_MTH = MTH（每一天所属的月份应该是当前日期所在月份）作为筛选条件。下面给出了内联视图 Y 的部分行：

```

select date_format(dy,'%u') wk,

```



```

        date_format(dy,'%d') dm,
        date_format(dy,'%w')+1 dw
    from (
select adddate(x.dy,t500.id-1) dy,
       x.mth
    from (
select adddate(current_date,-dayofmonth(current_date)+1) dy,
       date_format(
           adddate(current_date,
                    -dayofmonth(current_date)+1),
                    '%m') mth

    from t1
       ) x,
       t500
    where t500.id <= 31
      and date_format(adddate(x.dy,t500.id-1),'%m') = x.mth
       ) y

```

WK	DM	DW
--	--	-----
22	01	4
22	02	5
...		
25	21	3
25	22	4
...		
26	30	5

当前月的每一天包含下列信息：两位数字的月份日期（DM）、一位数字表示的星期几（DW）以及两位数字的ISO周序号（WK）。有了这些信息，就可以编写一个CASE表达式，以便确定DM（当前月的每一天）中每个值对应星期几。下面给出了部分结果：

```

select case dw when 2 then dm end as Mo,
       case dw when 3 then dm end as Tu,
       case dw when 4 then dm end as We,
       case dw when 5 then dm end as Th,
       case dw when 6 then dm end as Fr,
       case dw when 7 then dm end as Sa,
       case dw when 1 then dm end as Su
    from (
select date_format(dy,'%u') wk,
       date_format(dy,'%d') dm,
       date_format(dy,'%w')+1 dw
    from (
select adddate(x.dy,t500.id-1) dy,
       x.mth
    from (
select adddate(current_date,-dayofmonth(current_date)+1) dy,
       date_format(
           adddate(current_date,
                    -dayofmonth(current_date)+1),
                    '%m') mth

    from t1
       ) x,
       t500
    where t500.id <= 31
      and date_format(adddate(x.dy,t500.id-1),'%m') = x.mth
       ) y
       ) z

```

WK	MO	TU	WE	TH	FR	SA	SU
--	--	--	--	--	--	--	--
22			01				
22				02			
22					03		
22						04	

```

22          05
23 06          05
23    07
23      08
23        09
23          10
23            11
23              12
23

```

从上面的输出结果可以看出，对每周的每一天都返回一行信息。在每行内，月份日期就在与这天的周内日期相对应的列。现在需要做的是把每周的所有日期都放入一行中，要实现这种功能，可使用聚集函数 MAX，并按照 WK（ISO 周序号）分组。对结果按 WK 排序，以保证所有日期的顺序正确。最终输出结果如下：

```

select max(case dw when 2 then dm end) as Mo,
       max(case dw when 3 then dm end) as Tu,
       max(case dw when 4 then dm end) as We,
       max(case dw when 5 then dm end) as Th,
       max(case dw when 6 then dm end) as Fr,
       max(case dw when 7 then dm end) as Sa,
       max(case dw when 1 then dm end) as Su
  from (
select date_format(dy,'%u') wk,
       date_format(dy,'%d') dm,
       date_format(dy,'%w')+1 dw
  from (
select adddate(x.dy,t500.id-1) dy,
       x.mth
  from (
select adddate(current_date,-dayofmonth(current_date)+1) dy,
       date_format(
         adddate(current_date,
                  -dayofmonth(current_date)+1),
         '%m') mth
  from t1
  ) x,
  t500
 where t500.id <= 31
    and date_format(adddate(x.dy,t500.id-1),'%m') = x.mth
  ) y
  ) z
 group by wk
 order by wk

MO TU WE TH FR SA SU
-- -- -- -- -- -- --
      01 02 03 04 05
06 07 08 09 10 11 12
13 14 15 16 17 18 19
20 21 22 23 24 25 26
27 28 29 30

```

## SQL Server

首先，为当前月的每一天返回一行信息。使用递归 WITH 子句可实现该功能。如果使用的 SQL Server 版本不支持递归 WITH，则可以使用基干表，例如 MySQL 解决方案中的 T500。返回的每一行包含下列信息：月份日期（别名 DM）、星期几（别名 DW）、当前月份（别名 MTH）、ISO 周序号（别名 WK）。在递归之前，递归视图 X 产生的结果（UNION ALL 的上半部分）如下所示：

```

select dy,
       day(dy) dm,
       datepart(m,dy) mth,
       datepart(dw,dy) dw,
       case when datepart(dw,dy) = 1
            then datepart(dw,dy)-1
            else datepart(dw,dy)
       end wk
from (
select dateadd(day,-day(getdate()+1,getdate()) dy
from t1
) x

```

DY	DM	MTH	DW	WK
01-JUN-2005	1	6	4	23

下一步，重复递增DM值（递增次数就是月份天数），直到超出当前月为止。在对当前月的每一天进行处理时，也会得到每天对应星期几以及当日的ISO周序号，下面给出了部分结果：

```

with x(dy, dm, mth, dw, wk)
as (
select dy,
       day(dy) dm,
       datepart(m,dy) mth,
       datepart(dw,dy) dw,
       case when datepart(dw,dy) = 1
            then datepart(dw,dy)-1
            else datepart(dw,dy)
       end wk
from (
select dateadd(day,-day(getdate()+1,getdate()) dy
from t1
) x
union all
select dateadd(d,1,dy), day(dateadd(d,1,dy)), mth,
       datepart(dw,dateadd(d,1,dy)),
       case when datepart(dw,dateadd(d,1,dy)) = 1
            then datepart(dw,dateadd(d,1,dy))-1
            else datepart(dw,dateadd(d,1,dy))
       end
from x
where datepart(m,dateadd(d,1,dy)) = mth
)
select *
from x

```

DY	DM	MTH	DW	WK
01-JUN-2005	01	06	4	23
02-JUN-2005	02	06	5	23
...				
21-JUN-2005	21	06	3	26
22-JUN-2005	22	06	4	26
...				
30-JUN-2005	30	06	5	27

此时，当前月的每一天包含如下信息：两位数字的月份日期值、两位数字的月份值、一位数字表示的星期几（1~7 分别对应星期日~星期六）以及两位数字的ISO周序号。

现在，可以使用一个CASE表达式确定DM（当前月的每一天）中每个值对应星期几。下面给出了部分结果：

```

with x(dy, dm, mth, dw, wk)
as (
select dy,
      day(dy) dm,
      datepart(m, dy) mth,
      datepart(dw, dy) dw,
      case when datepart(dw, dy) = 1
            then datepart(wk, dy) - 1
            else datepart(wk, dy)
      end wk
from (
select dateadd(day, -day(getdate()) + 1, getdate()) dy
from t1
) x
union all
select dateadd(d, 1, dy), day(dateadd(d, 1, dy)), mth,
      datepart(dw, dateadd(d, 1, dy)),
      case when datepart(dw, dateadd(d, 1, dy)) = 1
            then datepart(wk, dateadd(d, 1, dy)) - 1
            else datepart(wk, dateadd(d, 1, dy))
      end
from x
where datepart(m, dateadd(d, 1, dy)) = mth
)
select case dw when 2 then dm end as Mo,
      case dw when 3 then dm end as Tu,
      case dw when 4 then dm end as We,
      case dw when 5 then dm end as Th,
      case dw when 6 then dm end as Fr,
      case dw when 7 then dm end as Sa,
      case dw when 1 then dm end as Su
from x

```

WK	MO	TU	WE	TH	FR	SA	SU
22			01				
22				02			
22					03		
22						04	
22							05
23	06						
23		07					
23			08				
23				09			
23					10		
23						11	
23							12

每周的每一天都独占一行。在每行中，包含日期编号的列都与星期名相对应。现在需要做的是把每周的所有日期都放入一行中。要实现这种功能，对各列使用聚集函数 MAX，并按 WK（ISO 周序号）给行分组。符合日历格式的结果如下：

```

with x(dy, dm, mth, dw, wk)
as (
select dy,
      day(dy) dm,
      datepart(m, dy) mth,
      datepart(dw, dy) dw,
      case when datepart(dw, dy) = 1
            then datepart(wk, dy) - 1
            else datepart(wk, dy)
      end wk
from (
select dateadd(day, -day(getdate()) + 1, getdate()) dy
from t1

```

```

) x
union all
select dateadd(d,1,dy), day(dateadd(d,1,dy)), mth,
       datepart(dw,dateadd(d,1,dy)),
       case when datepart(dw,dateadd(d,1,dy)) = 1
            then datepart(wk,dateadd(d,1,dy))-1
            else datepart(wk,dateadd(d,1,dy))
       end
       from x
       where datepart(m,dateadd(d,1,dy)) = mth
)
select max(case dw when 2 then dm end) as Mo,
       max(case dw when 3 then dm end) as Tu,
       max(case dw when 4 then dm end) as We,
       max(case dw when 5 then dm end) as Th,
       max(case dw when 6 then dm end) as Fr,
       max(case dw when 7 then dm end) as Sa,
       max(case dw when 1 then dm end) as Su
       from x
       group by wk
       order by wk

```

```

MO TU WE TH FR SA SU
-- -- -- -- -- -- --
      01 02 03 04 05
06 07 08 09 10 11 12
13 14 15 16 17 18 19
20 21 22 23 24 25 26
27 28 29 30

```

## 9.8 列出一年中每个季度的开始日期和结束日期问题

返回一年中每个季度的开始日期和结束日期。

### 解决方案

一年有4个季度，因此需要生成4行信息。在生成想要的行数之后，只需使用RDBMS提供的日期函数，就能返回开始日期和结束日期所属的季度。最后将产生下列结果集（跟前几节一样，选择使用当前年也是随意的）：

QTR	Q_START	Q_END
1	01-JAN-2005	31-MAR-2005
2	01-APR-2005	30-JUN-2005
3	01-JUL-2005	30-SEP-2005
4	01-OCT-2005	31-DEC-2005

### DB2

使用表EMP和窗口函数ROW\_NUMBER OVER，生成4行信息。另外，也可以采用WITH子句生成信息行（正如很多方案所讲的），还可以查询任意表（至少包含4行信息）。下列解决方案采用了ROW\_NUMBER OVER方法：

```

1 select quarter(dy-1 day) QTR,
2        dy-3 month Q_start,
3        dy-1 day Q_end
4 from (

```

```

5 select (current_date -
6         (dayofyear(current_date)-1) day
7         + (rn*3) month) dy
8   from (
9 select row_number()over() rn
10  from emp
11  fetch first 4 rows only
12      ) x
13      ) y

```

### Oracle

使用函数 ADD\_MONTHS，获得每个季度的开始日期和结束日期。使用 ROWNUM，表示开始日期和结束日期所属的季度。下面的解决方案使用表 EMP 生成 4 行信息。

```

1 select rownum qtr,
2        add_months(trunc(sysdate,'y'),(rownum-1)*3) q_start,
3        add_months(trunc(sysdate,'y'),rownum*3)-1 q_end
4   from emp
5  where rownum <= 4

```

### PostgreSQL

使用函数 GENERATE\_SERIES，生成 4 个季度。使用 DATE\_TRUNC 函数，把为每个季度生成的日期截断为年和月。使用 TO\_CHAR 函数，确定开始日期和结束日期所属的季度：

```

1 select to_char(dy,'Q') as QTR,
2        date(
3          date_trunc('month',dy)-(2*interval '1 month')
4        ) as Q_start,
5        dy as Q_end
6   from (
7 select date(dy+((rn*3) * interval '1 month'))-1 as dy
8   from (
9 select rn, date(date_trunc('year',current_date)) as dy
10    from generate_series(1,4) gs(rn)
11      ) x
12      ) y

```

### MySQL

使用表 T500 生成 4 行信息（每行表示一个季度）。使用函数 DATE\_ADD 和 ADDDATE，创建每个季度的开始日期和结束日期。使用 QUARTER 函数，确定开始日期和结束日期所属的季度：

```

1 select quarter(adddate(dy,-1)) QTR,
2        date_add(dy,interval -3 month) Q_start,
3        adddate(dy,-1) Q_end
4   from (
5 select date_add(dy,interval (3*id) month) dy
6   from (
7 select id,
8        adddate(current_date,-dayofyear(current_date)+1) dy
9   from t500
10  where id <= 4
11      ) x
12      ) y

```

## SQL Server

使用递归 WITH 子句生成 4 行信息。使用函数 DATEADD, 获得开始日期和结束日期。使用函数 DATEPART, 确定开始日期和结束日期所属的季度:

```

1  with x (dy,cnt)
2  as (
3  select dateadd(d,-(datepart(dy,getdate())-1),getdate()),
4         1
5  from t1
6  union all
7  select dateadd(m,3,dy), cnt+1
8  from x
9  where cnt+1 <= 4
10 )
11 select datepart(q,dateadd(d,-1,dy)) QTR,
12        dateadd(m,-3,dy) Q_start,
13        dateadd(d,-1,dy) Q_end
14 from x
15 order by 1

```

## 讨论

### DB2

首先, 为一年的 4 个季节生成 4 行信息 (它的值从 1 到 4)。内联视图 X 使用窗口函数 ROW\_NUMBER OVER 和 FETCH FIRST 子句, 返回来自 EMP 的 4 行信息。其结果如下:

```

select row_number()over() rn
from emp
fetch first 4 rows only

```

```

RN
--
1
2
3
4

```

下一步, 找到当前年的第一天, 然后加 n 个月, 其中 n 是 RN 的 3 倍 (把 3、6、9 和 12 与当前年的第一天相加)。其结果如下所示:

```

select (current_date -
        (dayofyear(current_date)-1) day
        + (rn*3) month) dy
from (
select row_number()over() rn
from emp
fetch first 4 rows only
) x

```

```

DY
-----
01-APR-2005
01-JUL-2005
01-OCT-2005
01-JAN-2006

```

此时, DY 的值是每个季度结束日期的后一天。下一步, 获取每个季度的开始日期和结束

日期。从DY中减去1天，就可以得到每个季度的结束日期，从DY中减去3个月，就可以得到每个季度的开始日期。对DY-1（每个季度的结束日期）使用QUARTER函数，以确定开始日期和结束日期所属的季度。

## Oracle

同时使用ROWNUM、TRUNC和ADD\_MONTHS，会使这种解决方案相当容易。要得到每个季度的开始日期，只需给当前年的第一天加n个月，其中 $n = (\text{ROWNUM}-1)*3$  (0,3,6,9)。要得到每个季度的结束日期，需给当前年的第一天加n个月，其中 $n = \text{ROWNUM}*3$ ，再减去1天。另外，如果对季度进行处理操作，使用带有'q'格式选项的TO\_CHAR和/或TRUNC会非常有用。

## PostgreSQL

首先，使用DATE\_TRUNC函数把当前日期截断为当前年的第一天。然后，加n个月，其中n是RN（由GENERATE\_SERIES返回的值）的3倍，再减去1天。其结果如下所示：

```
select date(dy+((rn*3) * interval '1 month'))-1 as dy
  from (
select rn, date(date_trunc('year',current_date)) as dy
  from generate_series(1,4) gs(rn)
) x
```

```
DY
-----
31-MAR-2005
30-JUN-2005
30-SEP-2005
31-DEC-2005
```

现在得到了每个季度的结束日期。最后一步，从DY中减去2个月，再使用DATE\_TRUNC函数截断到当前月的第一天，就能够获得开始日期。对每个季度的结束日期（DY）使用TO\_CHAR函数，以确定开始日期和结束日期所属的季度。

## MySQL

第一步，使用函数ADDDATE和DAYOFYEAR，找到当前年的第一天，然后使用DATE\_ADD函数，给当前年的第一天加n个月，其中n是T500.ID的3倍。其结果如下所示：

```
select date_add(dy,interval (3*id) month) dy
  from (
select id,
  adddate(current_date,-dayofyear(current_date)+1) dy
  from t500
 where id <= 4
) x
```

```
DY
-----
01-APR-2005
01-JUL-2005
01-OCT-2005
```



01-JAN-2005

此时，得到的日期是每个季度结束日期的后一天；要获取每个季度的结束日期，只需从 DY 中减去 1 天。然后，从 DY 中减去 3 个月，就可以得到每个季度的开始日期。对每个季度的结束日期使用 QUARTER 函数，可确定开始日期和结束日期所属的季度。

## SQL Server

首先找到当前年的第一天，然后用 DATEADD 函数，递归在当前年第一天上加 n 个月，其中 n 是当前迭代次数的 3 倍（共迭代 4 次，因此实际上是分别加 3\*1 个月、3\*2 个月，等等）。其结果如下所示：

```
with x (dy,cnt)
as (
select dateadd(d,-(datepart(dy,getdate())-1),getdate()),
       1
  from t1
 union all
select dateadd(m,3,dy), cnt+1
  from x
 where cnt+1 <= 4
)
select dy
  from x

DY
-----
01-APR-2005
01-JUL-2005
01-OCT-2005
01-JAN-2005
```

此时，DY 的值是每个季度结束日期的后一天；要获取每个季度的结束日期，只需使用 DATEADD 函数，只是从 DY 中减去 1 天；然后，使用 DATEADD 函数，从 DY 中减去 3 个月，就可以得到每个季度的开始日期。对每个季度的结束日期用 DATEPART 函数，可确定开始日期和结束日期所属的季度。

## 9.9 确定某个给定季度的开始日期和结束日期

### 问题

对于 YYYYQ 格式（其中 4 位年、一位季度）的年 and 季度信息，返回该季度的开始日期和结束日期。

### 解决方案

这个解决方案的关键是对 YYYYQ 值用求模函数得到季度值（由于年格式是 4 位数字，所以还可以用另一种方法得到季度值，即提取最后一位数字）。获得季度值之后乘以 3，就能得到该季度的结束月份。在下面的解决方案中，内联视图 X 将返回 4 行年和季度的综合信息。内联视图 X 的结果集如下：

```
select 20051 as yrq from t1 union all
select 20052 as yrq from t1 union all
select 20053 as yrq from t1 union all
select 20054 as yrq from t1
```

```
      YRQ
-----
20051
20052
20053
20054
```

## DB2

使用函数 SUBSTR，返回内联视图 X 中的年。使用 MOD 函数确定要查询的季度：

```
1 select (q_end-2 month) q_start,
2        (q_end+1 month)-1 day q_end
3   from (
4 select date(substr(cast(yrq as char(4)),1,4) || '-' ||
5        rtrim(cast(mod(yrq,10)*3 as char(2))) || '-1') q_end
6   from (
7 select 20051 yrq from t1 union all
8 select 20052 yrq from t1 union all
9 select 20053 yrq from t1 union all
10 select 20054 yrq from t1
11        ) x
12        ) y
```

## Oracle

使用函数 SUBSTR，返回内联视图 X 中的年。使用 MOD 函数确定要查询的季度：

```
1 select add_months(q_end,-2) q_start,
2        last_day(q_end) q_end
3   from (
4 select to_date(substr(yrq,1,4)||mod(yrq,10)*3,'yyyymm') q_end
5   from (
6 select 20051 yrq from dual union all
7 select 20052 yrq from dual union all
8 select 20053 yrq from dual union all
9 select 20054 yrq from dual
10        ) x
11        ) y
```

## PostgreSQL

使用函数 SUBSTR，返回内联视图 X 中的年。使用 MOD 函数确定要查询的季度：

```
1 select date(q_end-(2*interval '1 month')) as q_start,
2        date(q_end+interval '1 month'-interval '1 day') as q_end
3   from (
4 select to_date(substr(yrq,1,4)||mod(yrq,10)*3,'yyyymm') as q_end
5   from (
6 select 20051 as yrq from t1 union all
7 select 20052 as yrq from t1 union all
8 select 20053 as yrq from t1 union all
9 select 20054 as yrq from t1
10        ) x
11        ) y
```

## MySQL

使用函数 SUBSTR，返回内联视图 X 中的年。使用 MOD 函数确定要查询的季度：

```

1 select date_add(
2     adddate(q_end,-day(q_end)+1),
3     interval -2 month) q_start,
4     q_end
5 from (
6 select last_day(
7     str_to_date(
8         concat(
9             substr(yrq,1,4),mod(yrq,10)*3),'%Y%m')) q_end
10 from (
11 select 20051 as yrq from t1 union all
12 select 20052 as yrq from t1 union all
13 select 20053 as yrq from t1 union all
14 select 20054 as yrq from t1
15 ) x
16 ) y

```

## SQL Server

使用函数SUBSTRING, 返回内联视图X中的年。使用模数函数(%) 确定要查询的季度:

```

1 select dateadd(m,-2,q_end) q_start,
2     dateadd(d,-1,dateadd(m,1,q_end)) q_end
3 from (
4 select cast(substring(cast(yrq as varchar),1,4)+'-'+
5     cast(yrq%10*3 as varchar)+'-1' as datetime) q_end
6 from (
7 select 20051 yrq from t1 union all
8 select 20052 yrq from t1 union all
9 select 20053 yrq from t1 union all
10 select 20054 yrq from t1
11 ) x
12 ) y

```

## 讨论

### DB2

第一步, 先找到年和季度。使用SUBSTR 函数从内联视图 X(X.YRQ)中提取年子串。要获取季度值, 需求YRQ值对10的模。获得了季度值之后乘以3, 就能得到该季度的结束月份。其结果如下所示:

```

select substr(cast(yrq as char(4)),1,4) yr,
       mod(yrq,10)*3 mth
from (
select 20051 yrq from t1 union all
select 20052 yrq from t1 union all
select 20053 yrq from t1 union all
select 20054 yrq from t1
) x

```

YR	MTH
2005	3
2005	6
2005	9
2005	12

此时, 已获得了年及每个季度的结束月份。使用这些值可构造一个日期, 尤其是每个季度最后一个月的第一天。使用联接操作符“||”, 把年和月连接在一起, 然后使用DATE 函数, 把它转换为日期值:

```

select date(substr(cast(yrq as char(4)),1,4) || '-' ||
               rtrim(cast(mod(yrq,10)*3 as char(2))) || '-1') q_end
  from (
select 20051 yrq from t1 union all
select 20052 yrq from t1 union all
select 20053 yrq from t1 union all
select 20054 yrq from t1
  ) x

Q_END
-----
01-MAR-2005
01-JUN-2005
01-SEP-2005
01-DEC-2005

```

Q\_END值是每个季度最后一个月的第一天。要得到某个月份的最后一天,可以给Q\_END加1个月,然后减1天;要得到每个季度的开始日期,只需从Q\_END中减去2个月即可。

## Oracle

第一步,先找到年和季度。使用SUBSTR函数从内联视图X(X.YRQ)中提取年子串。要获取季度值,需求YRQ值对10的模。获得季度值之后乘以3,就能得到该季度的结束月份。其结果如下所示:

```

select substr(yrq,1,4) yr, mod(yrq,10)*3 mth
  from (
select 20051 yrq from dual union all
select 20052 yrq from dual union all
select 20053 yrq from dual union all
select 20054 yrq from dual
  ) x

YR      MTH
-----
2005      3
2005      6
2005      9
2005     12

```

此时,已获得了年及每个季度的结束月份。使用这些值可构造一个日期,尤其是每个季度最后一个月的第一天。使用联接操作符“||”,把年和月连接在一起,然后使用TO\_DATE函数,把它转换为日期值:

```

select to_date(substr(yrq,1,4)||mod(yrq,10)*3,'yyyymm') q_end
  from (
select 20051 yrq from dual union all
select 20052 yrq from dual union all
select 20053 yrq from dual union all
select 20054 yrq from dual
  ) x

Q_END
-----
01-MAR-2005
01-JUN-2005
01-SEP-2005
01-DEC-2005

```

Q\_END 值是每个季度最后一个月的第一天。要得到某个月份的最后一天,可使用

LAST\_DAY函数,要得到每个季度的开始日期,只需使用ADD\_MONTHS函数从Q\_END中减去2个月即可。

## PostgreSQL

第一步,先找到年和季度。使用SUBSTR函数从内联视图X(X.YRQ)中提取年子串;要获取季度值,需求YRQ值对10的模。获得季度值之后乘以3,就能得到该季度的结束月份。其结果如下所示:

```
select substr(yrq,1,4) yr, mod(yrq,10)*3 mth
  from (
select 20051 yrq from dual union all
select 20052 yrq from dual union all
select 20053 yrq from dual union all
select 20054 yrq from dual
  ) x
```

YR	MTH
2005	3
2005	6
2005	9
2005	12

此时,已获得了年及每个季度的结束月份。使用这些值可构造一个日期,尤其是每个季度最后一个月的第一天。使用联接操作符“||”,把年和月连接在一起,然后使用TO\_DATE函数,把它转换为日期值:

```
select to_date(substr(yrq,1,4)||mod(yrq,10)*3,'yyyymm') q_end
  from (
select 20051 yrq from dual union all
select 20052 yrq from dual union all
select 20053 yrq from dual union all
select 20054 yrq from dual
  ) x
```

Q_END
01-MAR-2005
01-JUN-2005
01-SEP-2005
01-DEC-2005

Q\_END值是每个季度最后一个月的第一天。要得到某个月份的最后一天,可以给Q\_END加1个月,然后减1天;要得到每个季度的开始日期,只需从Q\_END中减去2个月即可。

把最终结果转换为日期类型。

## MySQL

第一步,先找到年和季度。使用SUBSTR函数从内联视图X(X.YRQ)中提取年的子串。要获取季度值,需求YRQ值对10的模。获得季度值之后乘以3,就能得到该季度的结束月份。其结果如下所示:

```
select substr(yrq,1,4) yr, mod(yrq,10)*3 mth
  from (
```

```
select 20051 yrq from dual union all
select 20052 yrq from dual union all
select 20053 yrq from dual union all
select 20054 yrq from dual
) x
```

YR	MTH
2005	3
2005	6
2005	9
2005	12

此时，已获得了年及每个季度的结束月份。使用这些值可构造一个日期，尤其是每个季度最后一个月的最后一天。使用 CONCAT 函数，把年和月连接在一起，然后，使用 STR\_TO\_DATE 函数，把它转换为日期值，再使用 LAST\_DAY 函数找到每个季度的最后一天：

```
select last_day(
  str_to_date(
    concat(
      substr(yrq,1,4),mod(yrq,10)*3,'%Y%m')
    ) q_end
  from (
    select 20051 as yrq from t1 union all
    select 20052 as yrq from t1 union all
    select 20053 as yrq from t1 union all
    select 20054 as yrq from t1
  ) x
) Q_END
```

Q_END
31-MAR-2005
30-JUN-2005
30-SEP-2005
31-DEC-2005

得到了每个季度的结束日期后，接下来就应该获取每个季度的起始日期。使用 DAY 函数，可返回每个季度结束日期所在月份的月份日期，并使用 ADDBDATE 函数从 Q\_END 中减去这个值，就能得到前一个月的结束日期；加 1 天，可得到每个季度最后一个月的第一天。最后一步，使用 DATE\_ADD 函数，从每个季度最后一个月的第一天中减去 2 个月，就可获得每个季度的起始日期。

## SQL Server

第一步，先找到年和季度。使用 SUBSTRING 函数从内联视图 X (X.YRQ) 中提取年子串。要获取季度值，需求 YRQ 值对 10 的模。获得季度值之后乘以 3，就能得到该季度的结束月份。其结果如下所示：

```
select substring(yrq,1,4) yr, yrq%10*3 mth
  from (
    select 20051 yrq from dual union all
    select 20052 yrq from dual union all
    select 20053 yrq from dual union all
    select 20054 yrq from dual
  ) x
```

YR	MTH
2005	3

```

2005      6
2005      9
2005     12

```

此时，已获得了年及每个季度的结束月份。使用这些值可构造一个日期，尤其是每个季度最后一个月的第一天。使用联接操作符“+”，把年和月连接在一起，然后使用 CAST 函数，把它转换为日期值：

```

select cast(substring(cast(yrq as varchar),1,4)+'-' +
              cast(yrq%10*3 as varchar)+'-1' as datetime) q_end
  from (
select 20051 yrq from t1 union all
select 20052 yrq from t1 union all
select 20053 yrq from t1 union all
select 20054 yrq from t1
  ) x

Q_END
-----
01-MAR-2005
01-JUN-2005
01-SEP-2005
01-DEC-2005

```

Q\_END 值是每个季度最后一个月的第一天。要得到某个月份的最后一天，可以给 Q\_END 加 1 个月，然后使用 DATEADD 函数减 1 天；要得到每个季度的开始日期，只需使用 DATEADD 函数从 Q\_END 中减去 2 个月即可。

## 9.10 填充丢失的日期

### 问题

为给定范围内的每个日期（每个月、周或年）生成一行信息。这样的行集通常用于生成综合报告。例如，计算每年内每个月聘用的员工数。检查已聘用的所有员工的聘用日期，其范围为 1980—1983：

```

select distinct
      extract(year from hiredate) as year
  from emp

YEAR
-----
1980
1981
1982
1983

```

现在，要确定 1980—1983 年间每个月聘用的员工数。下面列出了希望得到得结果集中的部分：

MTH	NUM_HIRED
01-JAN-1981	0
01-FEB-1981	2
01-MAR-1981	0
01-APR-1981	1
01-MAY-1981	1

01-JUN-1981	1
01-JUL-1981	0
01-AUG-1981	0
01-SEP-1981	2
01-OCT-1981	0
01-NOV-1981	1
01-DEC-1981	2

## 解决方案

这里的窍门是为每个月返回一行信息,即使并未聘用任何员工(此时该数为0)。由于1980和1983年间并不是每个月都聘用过员工,因此必须生成这些月份,然后把按HIREDATE(聘用日期)与表EMP进行外联接(对HIREDATE截取月份,以便它能与生成的月份相匹配)。

### DB2

使用递归 WITH 子句生成每个月(从1980和1983年1月1日至1983年12月1日间每个月的第一天)。一旦获得了给定日期范围内的所有月份,则可以对表EMP进行外联接,并使用聚集函数 COUNT 计算每个月的聘用员工数:

```

1  with x (start_date,end_date)
2  as (
3  select (min(hiredate) -
4         dayofyear(min(hiredate)) day +1 day) start_date,
5         (max(hiredate) -
6         dayofyear(max(hiredate)) day +1 day) +1 year end_date
7  from emp
8  union all
9  select start_date +1 month, end_date
10 from x
11 where (start_date +1 month) < end_date
12 )
13 select x.start_date mth, count(e.hiredate) num_hired
14 from x left join emp e
15 on (x.start_date = (e.hiredate-(day(hiredate)-1) day))
16 group by x.start_date
17 order by 1

```

### Oracle

使用 CONNECT BY 子句,生成1980-1983年间的每个月。然后对表EMP进行外联接,并使用聚集函数 COUNT,计算每个月聘用的人数。对于 Oracle8i Database 及较早版本,不能使用 ANSI 外联接,也不能使用 CONNECT BY 生成行;简单方法是使用一个传统的基干表(就像 MySQL 解决方案中所使用的)。下面给出了一种使用 Oracle 的外联接语法,用于 Oracle 的解决方案:

```

1  with x
2  as (
3  select add_months(start_date,level-1) start_date
4  from (
5  select min(trunc(hiredate,'y')) start_date,
6         add_months(max(trunc(hiredate,'y')),12) end_date
7  from emp
8  )
9  connect by level <= months_between(end_date,start_date)

```



```

10 )
11 select x.start_date MTH, count(e.hiredate) num_hired
12   from x, emp e
13  where x.start_date = trunc(e.hiredate(+), 'mm')
14  group by x.start_date
15  order by 1

```

下面给出了 Oracle 的第二种解决方案，这次使用了 ANSI 语法：

```

1   with x
2   as (
3   select add_months(start_date, level-1) start_date
4     from (
5   select min(trunc(hiredate, 'y')) start_date,
6          add_months(max(trunc(hiredate, 'y')), 12) end_date
7     from emp
8    )
9   connect by level <= months_between(end_date, start_date)
10  )
11 select x.start_date MTH, count(e.hiredate) num_hired
12   from x left join emp e
13    on (x.start_date = trunc(e.hiredate, 'mm'))
14  group by x.start_date
15  order by 1

```

## PostgreSQL

为了提高可读性，该解决方案使用视图 V，返回聘用第一个员工那年的第一个月的第一天与聘用最新员工那年的第一个月的第一天之间的月份数差。把视图 V 返回的值作为 GENERATE\_SERIES 的第二个参数传递给它，这样所生成月份（行）的数目就是正确的。一旦获得了给定日期范围内的所有月份，则可以对表 EMP 进行外联接，并使用聚集函数 COUNT 计算每个月聘用的员工数：

```

create view v
as
select cast(
    extract(year from age(last_month, first_month)) * 12 - 1
    as integer) as mths
  from (
select cast(date_trunc('year', min(hiredate)) as date) as first_month,
       cast(cast(date_trunc('year', max(hiredate))
                as date) + interval '1 year'
                as date) as last_month
  from emp
 ) x

1 select y.mth, count(e.hiredate) as num_hired
2   from (
3   select cast(e.start_date + (x.id * interval '1 month')
4     as date) as mth
5     from generate_series (0, (select mths from v)) x(id),
6          ( select cast(
7             date_trunc('year', min(hiredate))
8             as date) as start_date
9           from emp ) e
10    ) y left join emp e
11     on (y.mth = date_trunc('month', e.hiredate))
12  group by y.mth
13  order by 1

```

## MySQL

使用基于表 T500 生成 1980–1983 年间的每个月。然后对表 EMP 进行外联接，并使用聚集函数 COUNT 计算每个月聘用的员工数：

```

1  select z.mth, count(e.hiredate) num_hired
2    from (
3  select date_add(min_hd,interval t500.id-1 month) mth
4    from (
5  select min_hd, date_add(max_hd,interval 11 month) max_hd
6    from (
7  select adddate(min(hiredate),-dayofyear(min(hiredate))+1) min_hd,
8         adddate(max(hiredate),-dayofyear(max(hiredate))+1) max_hd
9    from emp
10   ) x
11   ) y,
12   t500
13  where date_add(min_hd,interval t500.id-1 month) <= max_hd
14        ) z left join emp e
15        on (z.mth = adddate(
16                date_add(
17                last_day(e.hiredate),interval -1 month),1))
18  group by z.mth
19  order by 1

```

## SQL Server

使用递归 WITH 子句生成每个月（从 1980 和 1983 年 1 月 1 日至 1983 年 12 月 1 日间每个月的第一天）。一旦获得了给定日期范围内的所有月份，则可以对表 EMP 进行外联接，并使用聚集函数 COUNT 计算每个月聘用的员工数：

```

1  with x (start_date,end_date)
2    as (
3  select (min(hiredate) -
4         datepart(dy,min(hiredate))+1) start_date,
5         dateadd(yy,1,
6                (max(hiredate) -
7                 datepart(dy,max(hiredate))+1)) end_date
8    from emp
9   union all
10  select dateadd(mm,1,start_date), end_date
11    from x
12   where dateadd(mm,1,start_date) < end_date
13  )
14  select x.start_date mth, count(e.hiredate) num_hired
15    from x left join emp e
16    on (x.start_date =
17        dateadd(dd,-day(e.hiredate)+1,e.hiredate))
18  group by x.start_date
19  order by 1

```

## 讨论

### DB2

第一步，生成 1980–1983 年间的每个月（实际上是每个月的第一天）。先使用 DAYOFYEAR 函数分别求 HIREDATE 的最小值 (MIN) 和最大值 (MAX) 的年月，以便找到边界月份：

```

select (min(hiredate) -
        dayofyear(min(hiredate)) day +1 day) start_date,
       (max(hiredate) -
        dayofyear(max(hiredate)) day +1 day) +1 year end_date
  from emp
START_DATE  END_DATE
-----
01-JAN-1980 01-JAN-1984

```

然后，重复给 START\_DATE 加月份，以返回最后结果集需要包含的所有月份。END\_DATE 的值比实际值多 1 天，这没问题，因为在用递归方式给 START\_DATE 加月份时，可以在 END\_DATE 之前停止。已创建的部分月份如下所示：

```

with x (start_date,end_date)
as (
select (min(hiredate) -
        dayofyear(min(hiredate)) day +1 day) start_date,
       (max(hiredate) -
        dayofyear(max(hiredate)) day +1 day) +1 year end_date
  from emp
 union all
select start_date +1 month, end_date
  from x
 where (start_date +1 month) < end_date
)
select *
  from x
START_DATE  END_DATE
-----
01-JAN-1980 01-JAN-1984
01-FEB-1980 01-JAN-1984
01-MAR-1980 01-JAN-1984
...
01-OCT-1983 01-JAN-1984
01-NOV-1983 01-JAN-1984
01-DEC-1983 01-JAN-1984

```

此时，已经获取了所有月份，现在只需按 EMP.HIREDATE 进行外联接。由于每个 START\_DATE 的值都是当前月份第一天，故需对 EMP.HIREDATE 截取到当月的第一天。最后，用聚集函数 COUNT 对 EMP.HIREDATE 计数。

## Oracle

第一步，生成 1980–1983 年间的每个月的第一天。先使用 TRUNC、ADD\_MONTHS 函数分别求 HIREDATE 的最小值 (MIN) 和最大值 (MAX) 的年月，以便找到边界月份：

```

select min(trunc(hiredate,'y')) start_date,
       add_months(max(trunc(hiredate,'y')),12) end_date
  from emp
START_DATE  END_DATE
-----
01-JAN-1980 01-JAN-1984

```

然后，重复给 START\_DATE 加月份，以返回最后结果集所要包含的所有月份。END\_DATE 的值比实际值多 1 天，这没问题，因为在用递归方式给 START\_DATE 加月份时，可以在 END\_DATE 之前停止。已创建的部分月份如下：

```

with x as (
select add_months(start_date,level-1) start_date
  from (
select min(trunc(hiredate,'y')) start_date,
       add_months(max(trunc(hiredate,'y')),12) end_date
  from emp
  )
 connect by level <= months_between(end_date,start_date)
)
select *
  from x

START_DATE
-----
01-JAN-1980
01-FEB-1980
01-MAR-1980
...
01-OCT-1983
01-NOV-1983
01-DEC-1983

```

此时，已经获取了所有月份，那么现在只需按 EMP.HIREDATE 进行外联接。由于每个 START\_DATE 的值都是当前月份第一天，故需将 EMP.HIREDATE 截取到当月第一天。最后，用聚集函数 COUNT 对 EMP.HIREDATE 计数。

## PostgreSQL

该解决方案使用函数 GENERATE\_SERIES 返回想要的月份。如果不能使用 GENERATE\_SERIES 函数，则可以使用基于表，这在 MySQL 解决方案中曾介绍过。视图 V 只是通过查找给定范围内的边界日期而生成想要的月份数。视图 V 中的内联视图 X 用 MIN 和 MAX 函数根据 HIREDATE 找到开始和结束的边界日期，如下所示：

```

select cast(date_trunc('year',min(hiredate)) as date) as first_month,
       cast(cast(date_trunc('year',max(hiredate))
                as date) + interval '1 year'
                as date) as last_month
  from emp

FIRST_MONTH LAST_MONTH
-----
01-JAN-1980 01-JAN-1984

```

LAST\_MONTH 的值比实际值多 1，这没问题，因为在计算两个日期之间相差的月份数时可以再减 1。然后，使用 AGE 函数，查找两个日期之间相差的年数，再乘以 12（记住，要减 1）：

```

select cast(
       extract(year from age(last_month,first_month))*12-1
       as integer) as mths
  from (
select cast(date_trunc('year',min(hiredate)) as date) as first_month,
       cast(cast(date_trunc('year',max(hiredate))
                as date) + interval '1 year'
                as date) as last_month
  from emp
  ) x

```

```
MTHS
-----
47
```

把视图 V 返回的值作为 GENERATE\_SERIES 的第二个参数, 就可以返回想要的月份数。下一步将获取开始日期。重复给开始日期加月份值, 以创建要求范围内的所有月份。内联视图 Y 把 MIN(HIREDATE) 作为 DATE\_TRUNC 函数的参数, 以便找到开始日期, 并把 GENERATE\_SERIES 返回的值作为加到开始日期上的月份数。下面列出了部分返回结果:

```
select cast(e.start_date + (x.id * interval '1 month')
as date) as mth
from generate_series (0,(select mths from v)) x(id),
( select cast(
date_trunc('year',min(hiredate))
as date) as start_date
from emp
) e
```

```
MTH
-----
01-JAN-1980
01-FEB-1980
01-MAR-1980
...
01-OCT-1983
01-NOV-1983
01-DEC-1983
```

此时, 已经获取了最终结果集需要的所有月份, 现在只需按 EMP.HIREDATE 进行外联接, 再使用聚集函数 COUNT 计算每个月聘用的员工数。

## MySQL

首先, 使用聚集函数 MIN、MAX、DAYOFYEAR 和 ADDBDATE 函数, 获取边界日期。内联视图 X 中的结果集如下所示:

```
select adddate(min(hiredate),-dayofyear(min(hiredate))+1) min_hd,
addate(max(hiredate),-dayofyear(max(hiredate))+1) max_hd
from emp
```

```
MIN_HD      MAX_HD
-----
01-JAN-1980 01-JAN-1983
```

然后, 递增 MAX\_HD 值, 直到最后一个月:

```
select min_hd, date_add(max_hd,interval 11 month) max_hd
from (
select adddate(min(hiredate),-dayofyear(min(hiredate))+1) min_hd,
addate(max(hiredate),-dayofyear(max(hiredate))+1) max_hd
from emp
) x
```

```
MIN_HD      MAX_HD
-----
01-JAN-1980 01-DEC-1983
```

此时已经获取了边界日期。现在, 可使用基于表 T500, 给 MIN\_HD 加月份值, 直到与 MAX\_HD 相匹配为止, 这样就能够生成想要的所有行。下面给出了部分结果:

```

select date_add(min_hd,interval t500.id-1 month) mth
  from (
select min_hd, date_add(max_hd,interval 11 month) max_hd
  from (
select adddate(min(hiredate),-dayofyear(min(hiredate))+1) min_hd,
       adddate(max(hiredate),-dayofyear(max(hiredate))+1) max_hd
  from emp
   ) x
   ) y,
   t500
 where date_add(min_hd,interval t500.id-1 month) <= max_hd

```

MTH

```

-----
01-JAN-1980
01-FEB-1980
01-MAR-1980
...
01-OCT-1983
01-NOV-1983
01-DEC-1983

```

此时，已经获取了最终结果集需要的所有月份，现在只需按 EMP.HIREDATE 进行外联接（确保对 EMP.HIREDATE 截取到当月的第一天）。再用聚集函数 COUNT 对 EMP.HIREDATE 计数，得到每个月聘用的员工数。

## SQL Server

第一步，生成 1980-1983 年间的每个月（实际上是每个月的第一天）。先使用 DAYOFYEAR 函数分别求 HIREDATE 的最小值（MIN）和最大值（MAX）的年月，以便找到边界月份：

```

select (min(hiredate) -
       datepart(dy,min(hiredate))+1) start_date,
       dateadd(yy,1,
       (max(hiredate) -
       datepart(dy,max(hiredate))+1)) end_date
  from emp

START_DATE  END_DATE
-----
01-JAN-1980 01-JAN-1984

```

然后，重复给 START\_DATE 加月份，以返回最后结果集包含的所有月份。END\_DATE 的值比实际值多 1 天，这没问题，因为在用递归方式给 START\_DATE 加月份时，可以在 END\_DATE 之前停止。已创建的部分月份如下所示：

```

with x (start_date,end_date)
as (
select (min(hiredate) -
       datepart(dy,min(hiredate))+1) start_date,
       dateadd(yy,1,
       (max(hiredate) -
       datepart(dy,max(hiredate))+1)) end_date
  from emp
 union all
select dateadd(mm,1,start_date), end_date
  from x
 where dateadd(mm,1,start_date) < end_date

```

```
)
select *
  from x

START_DATE  END_DATE
-----
01-JAN-1980 01-JAN-1984
01-FEB-1980 01-JAN-1984
01-MAR-1980 01-JAN-1984
...
01-OCT-1983 01-JAN-1984
01-NOV-1983 01-JAN-1984
01-DEC-1983 01-JAN-1984
```

此时，已经获取了所有月份，现在只需按 EMP.HIREDATE 进行外联接。由于每个 START\_DATE 的值都是当前月的第一天，故需对 EMP.HIREDATE 截取到当月的第一天。最后，用聚集函数 COUNT 对 EMP.HIREDATE 计数。

## 9.11 按照给定的时间单位进行查找问题

查找与给定月份、星期几或其他时间单位相匹配的日期。例如，找到 2 月份和 12 月份聘用的所有员工，或者查找星期二聘用的所有员工。

### 解决方案

使用 RDBMS 提供的函数，按给定的月份名或星期几查找相应的日期。在各种场合中，本方案都是非常有用的。如果想要查找 HIREDATE，而且只想提取月份，过滤掉年份（或者对 HIREDATE 的其他部分感兴趣），则可以使用本方案。本节对这个问题的示例解决方案是按月份名和星期几进行查找。研究一下 RDBMS 的日期格式函数，很容易就可以修改这些解决方案，实现按年份、季度、年和季度的组合、月份和年份的组合等方式进行查找。

### DB2 和 MySQL

使用函数 MONTHNAME 和 DAYNAME，分别找到聘用员工的月份名、星期几：

```
1 select ename
2   from emp
3  where monthname(hiredate) in ('February', 'December')
4         or dayname(hiredate) = 'Tuesday'
```

### Oracle 和 PostgreSQL

使用函数 TO\_CHAR，找到聘用员工的月份名以及是星期几。使用函数 RTRIM，删除尾空：

```
1 select ename
2   from emp
3  where rtrim(to_char(hiredate, 'month')) in ('february', 'december')
4         or rtrim(to_char(hiredate, 'day')) = 'tuesday'
```

## SQL Server

使用函数 DATENAME, 找到聘用员工的月份名以及是星期几。

```
1 select ename
2   from emp
3  where datename(m,hiredate) in ('February','December')
4       or datename(dw,hiredate) = 'Tuesday'
```

## 讨论

每个解决方案的关键是要知道使用哪些函数以及如何使用它们。要检验这些函数的返回值, 可以将它们放入 SELECT 子句中, 并查看输出结果。下面列出了 DEPTNO 10 中有关员工的结果集 (这里使用了 SQL Server 语法):

```
select ename,datename(m,hiredate) mth,datename(dw,hiredate) dw
  from emp
 where deptno = 10
```

ENAME	MTH	DW
CLARK	June	Tuesday
KING	November	Tuesday
MILLER	January	Saturday

知道了函数返回的内容之后, 使用每个解决方案中介绍的函数来获得结果是相当容易的。

## 9.12 使用日期的特殊部分比较记录

### 问题

查找聘用日期月份和周内日期都相同的员工。例如, 如果在 1988 年 3 月 10 日星期一聘用了某个员工, 而在 2001 年 3 月 2 日星期一聘用了另一个员工, 那么, 由于二者的聘用日期都在星期一, 而且月份名一致, 则可以认为它们相匹配。在表 EMP 中, 只有 3 个员工满足这种需求。返回下列结果集:

```
MSG
-----
JAMES was hired on the same month and weekday as FORD
SCOTT was hired on the same month and weekday as JAMES
SCOTT was hired on the same month and weekday as FORD
```

### 解决方案

因为要把一个员工的 HIREDATE 与另一个员工的 HIREDATE 相比较, 所以需要对表 EMP 进行自联接, 这样, 就可以对 HIREDATE 的每种组合进行比较, 然后从每个 HIREDATE 中提取它是星期几以及月份名, 并进行比较。

### DB2

在自联接表 EMP 之后, 可使用函数 DAYOFWEEK 返回表示星期几的数字值, 使用函数 MONTHNAME 返回月份名:



```
1 select a.ename ||  
2     ' was hired on the same month and weekday as ' ||  
3     b.ename msg  
4   from emp a, emp b  
5  where (dayofweek(a.hiredate),monthname(a.hiredate)) =  
6         (dayofweek(b.hiredate),monthname(b.hiredate))  
7         and a.empno < b.empno  
8  order by a.ename
```

## Oracle 和 PostgreSQL

在自联接表 EMP 之后，可使用 TO\_CHAR 函数，得到 HIREDATE 的周内日期名称和月份名：

```
1 select a.ename ||  
2     ' was hired on the same month and weekday as ' ||  
3     b.ename as msg  
4   from emp a, emp b  
5  where to_char(a.hiredate,'DMON') =  
6         to_char(b.hiredate,'DMON')  
7         and a.empno < b.empno  
8  order by a.ename
```

## MySQL

在自联接表 EMP 之后，可使用 DATE\_FORMAT 函数，得到 HIREDATE 的周内日期名称和月份名：

```
1 select concat(a.ename,  
2     ' was hired on the same month and weekday as ',  
3     b.ename) msg  
4   from emp a, emp b  
5  where date_format(a.hiredate,'%w%M') =  
6         date_format(b.hiredate,'%w%M')  
7         and a.empno < b.empno  
8  order by a.ename
```

## SQL Server

在自联接表 EMP 之后，可使用 DATENAME 函数，得到 HIREDATE 的周内日期名称和月份名：

```
1 select a.ename +  
2     ' was hired on the same month and weekday as '+  
3     b.ename msg  
4   from emp a, emp b  
5  where datename(dw,a.hiredate) = datename(dw,b.hiredate)  
6         and datename(m,a.hiredate) = datename(m,b.hiredate)  
7         and a.empno < b.empno  
8  order by a.ename
```

## 讨论

这些解决方案之间的唯一差别是：设置 HIREDATE 格式所使用的日期函数不同。在下面的讨论中，将采用 Oracle/PostgreSQL 解决方案（因为它的代码最短），但相应的解释也适用于其他解决方案。

第一步，自联接EMP，这样，每个员工都可以访问其他员工的HIREDATE。下面给出了查询结果（用SCOTT进行筛选）：

```
select a.ename as scott, a.hiredate as scott_hd,
       b.ename as other_emps, b.hiredate as other_hds
from emp a, emp b
where a.ename = 'SCOTT'
and a.empno != b.empno
```

SCOTT	SCOTT_HD	OTHER_EMPS	OTHER_HDS
SCOTT	09-DEC-1982	SMITH	17-DEC-1980
SCOTT	09-DEC-1982	ALLEN	20-FEB-1981
SCOTT	09-DEC-1982	WARD	22-FEB-1981
SCOTT	09-DEC-1982	JONES	02-APR-1981
SCOTT	09-DEC-1982	MARTIN	28-SEP-1981
SCOTT	09-DEC-1982	BLAKE	01-MAY-1981
SCOTT	09-DEC-1982	CLARK	09-JUN-1981
SCOTT	09-DEC-1982	KING	17-NOV-1981
SCOTT	09-DEC-1982	TURNER	08-SEP-1981
SCOTT	09-DEC-1982	ADAMS	12-JAN-1983
SCOTT	09-DEC-1982	JAMES	03-DEC-1981
SCOTT	09-DEC-1982	FORD	03-DEC-1981
SCOTT	09-DEC-1982	MILLER	23-JAN-1982

自联接表EMP之后，可以把SCOTT的HIREDATE与其他员工的HIREDATE相比较。用EMPNO进行筛选，这样就不会把SCOTT的HIREDATE当作一个OTHER\_HDS返回。然后，使用RDBMS的日期格式函数比较HIREDATE的周内日期名和月份，并且保留那些匹配的行：

```
select a.ename as emp1, a.hiredate as emp1_hd,
       b.ename as emp2, b.hiredate as emp2_hd
from emp a, emp b
where to_char(a.hiredate, 'DMON') =
      to_char(b.hiredate, 'DMON')
and a.empno != b.empno
order by 1
```

EMP1	EMP1_HD	EMP2	EMP2_HD
FORD	03-DEC-1981	SCOTT	09-DEC-1982
FORD	03-DEC-1981	JAMES	03-DEC-1981
JAMES	03-DEC-1981	SCOTT	09-DEC-1982
JAMES	03-DEC-1981	FORD	03-DEC-1981
SCOTT	09-DEC-1982	JAMES	03-DEC-1981
SCOTT	09-DEC-1982	FORD	03-DEC-1981

此时，HIREDATE完全匹配了，但结果集只包含6行信息，而不是本节“问题”部分提到的3行信息。多出几行的原因是用EMPNO进行筛选造成的。使用“不等于”操作符，不会筛选出对称信息。例如，第一行与FORD和SCOTT相匹配，最后一行与SCOTT和FORD相匹配。从技术上讲，结果集中的6行信息是准确的，但有冗余。可使用“小于”操作符去除冗余（去除这些HIREDATE会使中间查询与最终结果集更接近）：

```
select a.ename as emp1, b.ename as emp2
from emp a, emp b
where to_char(a.hiredate, 'DMON') =
      to_char(b.hiredate, 'DMON')
and a.empno < b.empno
order by 1
```

EMP1	EMP2
JAMES	FORD
SCOTT	JAMES
SCOTT	FORD

最后一步只是把结果集连接起来，以形成消息。

## 9.13 识别重叠的日期范围

### 问题

查找员工在老工程结束之前就开始新工程的所有实例。请参阅表 EMP\_PROJECT:

```
select *
  from emp_project
```

EMPNO	ENAME	PROJ_ID	PROJ_START	PROJ_END
7782	CLARK	1	16-JUN-2005	18-JUN-2005
7782	CLARK	4	19-JUN-2005	24-JUN-2005
7782	CLARK	7	22-JUN-2005	25-JUN-2005
7782	CLARK	10	25-JUN-2005	28-JUN-2005
7782	CLARK	13	28-JUN-2005	02-JUL-2005
7839	KING	2	17-JUN-2005	21-JUN-2005
7839	KING	8	23-JUN-2005	25-JUN-2005
7839	KING	14	29-JUN-2005	30-JUN-2005
7839	KING	11	26-JUN-2005	27-JUN-2005
7839	KING	5	20-JUN-2005	24-JUN-2005
7934	MILLER	3	18-JUN-2005	22-JUN-2005
7934	MILLER	12	27-JUN-2005	28-JUN-2005
7934	MILLER	15	30-JUN-2005	03-JUL-2005
7934	MILLER	9	24-JUN-2005	27-JUN-2005
7934	MILLER	6	21-JUN-2005	23-JUN-2005

观察一下员工 KING 的结果，会发现 KING 在完成 PROJ\_ID 5 之前开始了 PROJ\_ID 8，而且在完成 PROJ\_ID 2 之前又开始了 PROJ\_ID 5。最终应返回下列结果集：

EMPNO	ENAME	MSG
7782	CLARK	project 7 overlaps project 4
7782	CLARK	project 10 overlaps project 7
7782	CLARK	project 13 overlaps project 10
7839	KING	project 8 overlaps project 5
7839	KING	project 5 overlaps project 2
7934	MILLER	project 12 overlaps project 9
7934	MILLER	project 6 overlaps project 3

### 解决方案

这里的关键是：找到 PROJ\_START（新工程的开始日期）出现于另一个工程的 PROJ\_START 之后 PROJ\_END 日期之前的那些行。首先，应该把每个工程与另一个工程相比较（同一个员工）。对 EMP\_PROJECT 按员工进行自联接，得到每个员工任意两个工程的所有组合。要找到重叠的情况，只需找到一个 PROJ\_ID 的 PROJ\_START 介于另一个 PROJ\_ID 的 PROJ\_START 和 PROJ\_END 之间的所有行即可。

## DB2、PostgreSQL 和 Oracle

自联接 EMP\_PROJECT。然后使用联接操作符 “||” 构造用于说明重叠工程的消息：

```
1 select a.empno,a.ename,  
2       'project '||b.proj_id||  
3       ' overlaps project '||a.proj_id as msg  
4   from emp_project a,  
5        emp_project b  
6  where a.empno = b.empno  
7        and b.proj_start >= a.proj_start  
8        and b.proj_start <= a.proj_end  
9        and a.proj_id != b.proj_id
```

## MySQL

自联接 EMP\_PROJECT。然后使用 CONCAT 函数构造用于说明重叠工程的消息：

```
1 select a.empno,a.ename,  
2       concat('project ',b.proj_id,  
3       ' overlaps project ',a.proj_id) as msg  
4   from emp_project a,  
5        emp_project b  
6  where a.empno = b.empno  
7        and b.proj_start >= a.proj_start  
8        and b.proj_start <= a.proj_end  
9        and a.proj_id != b.proj_id
```

## SQL Server

自联接 EMP\_PROJECT。然后使用字符串连接操作符 “+” 构造用于说明重叠工程的消息：

```
1 select a.empno,a.ename,  
2       'project '+b.proj_id+  
3       ' overlaps project '+a.proj_id as msg  
4   from emp_project a,  
5        emp_project b  
6  where a.empno = b.empno  
7        and b.proj_start >= a.proj_start  
8        and b.proj_start <= a.proj_end  
9        and a.proj_id != b.proj_id
```

## 讨论

这些解决方案的唯一差别是字符串连接的方法不同，因此采用 DB2 语法的讨论就可以覆盖这 3 种解决方案。首先，自联接 EMP\_PROJECT，这样就可以在不同工程间比较 PROJ\_START 日期。员工 KING 的自联接输出如下所示，观察一下每个工程是如何“看见”其他工程的：

```
select a.ename,  
       a.proj_id as a_id,  
       a.proj_start as a_start,  
       a.proj_end as a_end,  
       b.proj_id as b_id,  
       b.proj_start as b_start  
  from emp_project a,  
       emp_project b  
 where a.ename = 'KING'  
       and a.empno = b.empno  
       and a.proj_id != b.proj_id
```

order by 2

ENAME	A_ID	A_START	A_END	B_ID	B_START
KING	2	17-JUN-2005	21-JUN-2005	8	23-JUN-2005
KING	2	17-JUN-2005	21-JUN-2005	14	29-JUN-2005
KING	2	17-JUN-2005	21-JUN-2005	11	26-JUN-2005
KING	2	17-JUN-2005	21-JUN-2005	5	20-JUN-2005
KING	5	20-JUN-2005	24-JUN-2005	2	17-JUN-2005
KING	5	20-JUN-2005	24-JUN-2005	8	23-JUN-2005
KING	5	20-JUN-2005	24-JUN-2005	11	26-JUN-2005
KING	5	20-JUN-2005	24-JUN-2005	14	29-JUN-2005
KING	8	23-JUN-2005	25-JUN-2005	2	17-JUN-2005
KING	8	23-JUN-2005	25-JUN-2005	14	29-JUN-2005
KING	8	23-JUN-2005	25-JUN-2005	5	20-JUN-2005
KING	8	23-JUN-2005	25-JUN-2005	11	26-JUN-2005
KING	11	26-JUN-2005	27-JUN-2005	2	17-JUN-2005
KING	11	26-JUN-2005	27-JUN-2005	8	23-JUN-2005
KING	11	26-JUN-2005	27-JUN-2005	14	29-JUN-2005
KING	11	26-JUN-2005	27-JUN-2005	5	20-JUN-2005
KING	14	29-JUN-2005	30-JUN-2005	2	17-JUN-2005
KING	14	29-JUN-2005	30-JUN-2005	8	23-JUN-2005
KING	14	29-JUN-2005	30-JUN-2005	5	20-JUN-2005
KING	14	29-JUN-2005	30-JUN-2005	11	26-JUN-2005

从上面的结果集可以看到，自联接之后，很容易找到重叠日期，只是返回B\_START介于A\_START和A\_END之间的那些行即可。该解决方案的第7行和第8行包含一个WHERE子句，如下：

```
and b.proj_start >= a.proj_start
and b.proj_start <= a.proj_end
```

它实现了该功能。在获得了想要的行之后，将返回值连接起来就能构造消息。

如果每个员工的最大工程数是固定的，Oracle用户可以使用窗口函数LEAD OVER代替自联接。如果自联接很昂贵的话（如果自联接需要的资源比LEAD OVER排序需要的资源更多），可考虑采用这种方式。例如，下面给出了使用LEAD OVER为员工KING构造消息的例子：

```
select empno,
       ename,
       proj_id,
       proj_start,
       proj_end,
       case
         when lead(proj_start,1)over(order by proj_start)
              between proj_start and proj_end
         then lead(proj_id)over(order by proj_start)
         when lead(proj_start,2)over(order by proj_start)
              between proj_start and proj_end
         then lead(proj_id)over(order by proj_start)
         when lead(proj_start,3)over(order by proj_start)
              between proj_start and proj_end
         then lead(proj_id)over(order by proj_start)
         when lead(proj_start,4)over(order by proj_start)
              between proj_start and proj_end
         then lead(proj_id)over(order by proj_start)
         end is_overlap
  from emp_project
 where ename = 'KING'
```

EMPNO	ENAME	PROJ_ID	PROJ_START	PROJ_END	IS_OVERLAP
7839	KING	2	17-JUN-2005	21-JUN-2005	5
7839	KING	5	20-JUN-2005	24-JUN-2005	8
7839	KING	8	23-JUN-2005	25-JUN-2005	
7839	KING	11	26-JUN-2005	27-JUN-2005	
7839	KING	14	29-JUN-2005	30-JUN-2005	

对于员工 KING 来说，由于最大工程数固定为 5，则可以使用 LEAD OVER 检查所有工程的日期，而无需进行自联接。之后，产生结果集就相当容易了，只需保留 IS\_OVERLAP 不是 NULL 的行即可：

```
select empno,ename,
       'project '||is_overlap||
       ' overlaps project '||proj_id msg
from (
select empno,
       ename,
       proj_id,
       proj_start,
       proj_end,
       case
         when lead(proj_start,1)over(order by proj_start)
           between proj_start and proj_end
         then lead(proj_id)over(order by proj_start)
         when lead(proj_start,2)over(order by proj_start)
           between proj_start and proj_end
         then lead(proj_id)over(order by proj_start)
         when lead(proj_start,3)over(order by proj_start)
           between proj_start and proj_end
         then lead(proj_id)over(order by proj_start)
         when lead(proj_start,4)over(order by proj_start)
           between proj_start and proj_end
         then lead(proj_id)over(order by proj_start)
         end is_overlap
       from emp_project
       where ename = 'KING'
       )
where is_overlap is not null

EMPNO ENAME MSG
-----
7839 KING project 5 overlaps project 2
7839 KING project 8 overlaps project 5
```

如果让该解决方案处理所有员工（而不仅仅是 KING），需要在 LEAD OVER 函数中按 ENAME 分区：

```
select empno,ename,
       'project '||is_overlap||
       ' overlaps project '||proj_id msg
from (
select empno,
       ename,
       proj_id,
       proj_start,
       proj_end,
       case
         when lead(proj_start,1)over(partition by ename
                                     order by proj_start)
           between proj_start and proj_end
         then lead(proj_id)over(partition by ename
```

```
                order by proj_start)
when lead(proj_start,2)over(partition by ename
                           order by proj_start)
   between proj_start and proj_end
then lead(proj_id)over(partition by ename
                      order by proj_start)
when lead(proj_start,3)over(partition by ename
                           order by proj_start)
   between proj_start and proj_end
then lead(proj_id)over(partition by ename
                      order by proj_start)
when lead(proj_start,4)over(partition by ename
                           order by proj_start)
   between proj_start and proj_end
then lead(proj_id)over(partition by ename
                      order by proj_start)
end is_overlap
from emp_project
)
where is_overlap is not null
```

EMPNO	ENAME	MSG
7782	CLARK	project 7 overlaps project 4
7782	CLARK	project 10 overlaps project 7
7782	CLARK	project 13 overlaps project 10
7839	KING	project 5 overlaps project 2
7839	KING	project 8 overlaps project 5
7934	MILLER	project 6 overlaps project 3
7934	MILLER	project 12 overlaps project 9



## 第 10 章

# 范围处理

本章将介绍有关范围的常见查询。在日常生活中，范围是很常见的。例如，工程都包含连续时段。在 SQL 中，经常需要搜索范围、生成范围，甚至处理基于范围的数据。这里介绍的查询比前几章的查询更深入些，但它们同样非常通用，并且能开始从中领悟到：如果充分发挥其优势，SQL 究竟能干些什么。

### 10.1 定位连续值的范围

#### 问题

确定哪些行表示连续工程的范围。下面给出了视图 V 的结果集，其中包含有关工程及其开始日期和结束日期的数据：

```
select *
  from V

PROJ_ID PROJ_START PROJ_END
-----
1 01-JAN-2005 02-JAN-2005
2 02-JAN-2005 03-JAN-2005
3 03-JAN-2005 04-JAN-2005
4 04-JAN-2005 05-JAN-2005
5 06-JAN-2005 07-JAN-2005
6 16-JAN-2005 17-JAN-2005
7 17-JAN-2005 18-JAN-2005
8 18-JAN-2005 19-JAN-2005
9 19-JAN-2005 20-JAN-2005
10 21-JAN-2005 22-JAN-2005
11 26-JAN-2005 27-JAN-2005
12 27-JAN-2005 28-JAN-2005
13 28-JAN-2005 29-JAN-2005
14 29-JAN-2005 30-JAN-2005
```

除第一行之外，每行的 PROJ\_START 都应该等于它前一行的 PROJ\_END（“前一行”定义为：当前行的 PROJ\_ID - 1）。检查视图 V 的前 5 行，PROJ\_ID 值为 1~3 的工程都属于同一“组”，其中每个 PROJ\_END 都等于它下一行的 PROJ\_START 值。要找到连续工程的日期范围，应该返回满足下述条件的所有行，即当前行的 PROJ\_END 值等于下一行



的 PROJ\_START。如果结果集中只有前 5 行，则只应返回前 3 行。最终结果集（使用视图 V 中包含的所有行，共 14 行）应该是：

PROJ_ID	PROJ_START	PROJ_END
1	01-JAN-2005	02-JAN-2005
2	02-JAN-2005	03-JAN-2005
3	03-JAN-2005	04-JAN-2005
6	16-JAN-2005	17-JAN-2005
7	17-JAN-2005	18-JAN-2005
8	18-JAN-2005	19-JAN-2005
11	26-JAN-2005	27-JAN-2005
12	27-JAN-2005	28-JAN-2005
13	28-JAN-2005	29-JAN-2005

结果集排除了 PROJ\_ID 值为 4、5、9、10 和 14 的行，原因是这些行的 PROJ\_END 与它下一行的 PROJ\_START 并不匹配。

## 解决方案

### DB2、MySQL、PostgreSQL 和 SQL Server

使用自联接查找包含连续值的行：

```
1 select v1.proj_id,
2       v1.proj_start,
3       v1.proj_end
4   from V v1, V v2
5  where v1.proj_end = v2.proj_start
```

### Oracle

上述解决方案对 Oracle 也有效。这里介绍另一种解决方案，即使用窗口函数 LEAD OVER 查看“下一行”的 BEGIN\_DATE，这样就可以避免使用自联接：

```
1 select proj_id,proj_start,proj_end
2   from (
3 select proj_id,proj_start,proj_end,
4       lead(proj_start)over(order by proj_id) next_proj_start
5   from V
6   )
7  where next_proj_start = proj_end
```

## 讨论

### DB2、MySQL、PostgreSQL 和 SQL Server

通过自联接视图，每一行都可以与返回的其他行相比较。下面列出了 ID 值为 1~4 的部分结果集：

```
select v1.proj_id as v1_id,
       v1.proj_end as v1_end,
       v2.proj_start as v2_begin,
       v2.proj_id as v2_id
  from v v1, v v2
 where v1.proj_id in ( 1,4 )
```

V1_ID	V1_END	V2_BEGIN	V2_ID
1	02-JAN-2005	01-JAN-2005	1
1	02-JAN-2005	02-JAN-2005	2
1	02-JAN-2005	03-JAN-2005	3
1	02-JAN-2005	04-JAN-2005	4
1	02-JAN-2005	06-JAN-2005	5
1	02-JAN-2005	16-JAN-2005	6
1	02-JAN-2005	17-JAN-2005	7
1	02-JAN-2005	18-JAN-2005	8
1	02-JAN-2005	19-JAN-2005	9
1	02-JAN-2005	21-JAN-2005	10
1	02-JAN-2005	26-JAN-2005	11
1	02-JAN-2005	27-JAN-2005	12
1	02-JAN-2005	28-JAN-2005	13
1	02-JAN-2005	29-JAN-2005	14
4	05-JAN-2005	01-JAN-2005	1
4	05-JAN-2005	02-JAN-2005	2
4	05-JAN-2005	03-JAN-2005	3
4	05-JAN-2005	04-JAN-2005	4
4	05-JAN-2005	06-JAN-2005	5
4	05-JAN-2005	16-JAN-2005	6
4	05-JAN-2005	17-JAN-2005	7
4	05-JAN-2005	18-JAN-2005	8
4	05-JAN-2005	19-JAN-2005	9
4	05-JAN-2005	21-JAN-2005	10
4	05-JAN-2005	26-JAN-2005	11
4	05-JAN-2005	27-JAN-2005	12
4	05-JAN-2005	28-JAN-2005	13
4	05-JAN-2005	29-JAN-2005	14

检查此结果集，会发现最终结果集包含了 PROJ\_ID 1，而排除了 PROJ\_ID 4，其原因是：为 V1\_ID 4 返回的 V1\_END 值并没有与之对应的 V2\_BEGIN 值。

根据查看数据的方式，把 PROJ\_ID 4 作为连续的看待也很容易。下面给出了结果集：

```
select *
  from v
 where proj_id <= 5
```

PROJ_ID	PROJ_START	PROJ_END
1	01-JAN-2005	02-JAN-2005
2	02-JAN-2005	03-JAN-2005
3	03-JAN-2005	04-JAN-2005
4	04-JAN-2005	05-JAN-2005
5	06-JAN-2005	07-JAN-2005

如果把“连续”定义为一个工程的开始日期与另一个工程的结束日期相同，那么结果集就应该包含 PROJ\_ID 4。最初删除 PROJ\_ID 4，是因为进行了向前比较（把它的 PROJ\_END 与下一个工程的 PROJ\_START 相比较），但如果进行向后比较（把它的 PROJ\_START 与前一个工程的 PROJ\_END 相比较），那么结果集就会引入 PROJ\_ID 4。

为包含 PROJ\_ID 4 而修改解决方案也非常简单：只需添加一个谓词，以确保在判断是否连续时既检查 PROJ\_START 又检查 PROJ\_END，而不仅仅检查 PROJ\_END。修改后的查询如下所示，它产生的结果集包含了 PROJ\_ID 4（一定要使用 DISTINCT 关键字，这是因为有些行同时满足两个谓词条件）：

```
select distinct
```

```

      v1.proj_id,
      v1.proj_start,
      v1.proj_end
    from V v1, V v2
    where v1.proj_end = v2.proj_start
      or v1.proj_start = v2.proj_end

```

PROJ_ID	PROJ_START	PROJ_END
1	01-JAN-2005	02-JAN-2005
2	02-JAN-2005	03-JAN-2005
3	03-JAN-2005	04-JAN-2005
4	04-JAN-2005	05-JAN-2005

## Oracle

尽管自联接解决方案也可以适用，然而用窗口函数 LEAD OVER 来解决这种类型的问题更完美。函数 LEAD OVER 可以检查其他行，而无需进行自联接（要实现这种功能，该函数必须给结果集排顺）。下面给出了有关 ID 值为 1 和 4 的内联视图（第 3~5 行）的结果：

```

select *
  from (
select proj_id,proj_start,proj_end,
       lead(proj_start)over(order by proj_id) next_proj_start
  from v
  )
 where proj_id in ( 1,4 )

```

PROJ_ID	PROJ_START	PROJ_END	NEXT_PROJ_START
1	01-JAN-2005	02-JAN-2005	02-JAN-2005
4	04-JAN-2005	05-JAN-2005	06-JAN-2005

检验上面的代码片段及其结果集，很容易会明白 PROJ\_ID 4 为何没有包含在最终结果集中。因为它的 PROJ\_END 日期为 05-JAN-2005，与“下一个”工程的开始日期 06-JAN-2005 并不匹配。

在遇到此类问题时，用函数 LEAD OVER 极其便利，尤其是检验部分结果时更是如此。当使用视窗函数时，应该记住，它们都在 FROM 和 WHERE 子句之后求值，因此，上述查询的 LEAD OVER 函数必须嵌入内联视图内，否则，LEAD OVER 函数只能作用于 WHERE 子句筛选过之后的结果集，就只剩 PROJ\_ID 为 1 和 4 的行。

现在，根据观察数据的方式，可能非常希望将 PROJ\_ID 4 包含到最终结果集中。看看下面视图 V 的前 5 行数据：

```

select *
  from v
 where proj_id <= 5

```

PROJ_ID	PROJ_START	PROJ_END
1	01-JAN-2005	02-JAN-2005
2	02-JAN-2005	03-JAN-2005
3	03-JAN-2005	04-JAN-2005
4	04-JAN-2005	05-JAN-2005
5	06-JAN-2005	07-JAN-2005

如果要求将 PROJ\_ID 4 也作为连续的（由于 PROJ\_ID 4 的 PROJ\_START 与 PROJ\_ID 3

的 PROJ\_END 相匹配), 而只有 PROJ\_ID 5 被排除在外, 那么本节的解决方案就不正确了 (!), 至少是不完善的:

```
select proj_id,proj_start,proj_end
  from (
select proj_id,proj_start,proj_end,
       lead(proj_start)over(order by proj_id) next_start
  from v
 where proj_id <= 5
 )
 where proj_end = next_start
```

PROJ_ID	PROJ_START	PROJ_END
1	01-JAN-2005	02-JAN-2005
2	02-JAN-2005	03-JAN-2005
3	03-JAN-2005	04-JAN-2005

如果确定应该包含 PROJ\_ID 4, 那么只需在查询中加入 LAG OVER, 并在 WHERE 子句中加一个筛选条件:

```
select proj_id,proj_start,proj_end
  from (
select proj_id,proj_start,proj_end,
       lead(proj_start)over(order by proj_id) next_start,
       lag(proj_end)over(order by proj_id) last_end
  from v
 where proj_id <= 5
 )
 where proj_end = next_start
       or proj_start = last_end
```

PROJ_ID	PROJ_START	PROJ_END
1	01-JAN-2005	02-JAN-2005
2	02-JAN-2005	03-JAN-2005
3	03-JAN-2005	04-JAN-2005
4	04-JAN-2005	05-JAN-2005

现在, 最终结果集中包含了 PROJ\_ID 4, 而且把不尽人意的 PROJ\_ID 5 排除了。实际应用中使用这些方案时, 应该考虑自己的确切需求。

## 10.2 查找同一组或分区中行之间的差问题

返回每个员工的 DEPTNO、ENAME 和 SAL 以及与同一部门 (即 DEPTNO 值相同) 的员工间的 SAL 之差, 该差值在当前员工及同部门内紧随其后聘用的员工间计算而来 (要从“每个部门”的角度看看资历和工资之间是否具有相关性)。对于每个部门中最新聘用的员工, 这个差值为 “N/A”。其结果集应该如下:

DEPTNO	ENAME	SAL	HIREDATE	DIFF
10	CLARK	2450	09-JUN-1981	-2550
10	KING	5000	17-NOV-1981	3700
10	MILLER	1300	23-JAN-1982	N/A
20	SMITH	800	17-DEC-1980	-2175
20	JONES	2975	02-APR-1981	-25
20	FORD	3000	03-DEC-1981	0

20	SCOTT	3000	09-DEC-1982	1900
20	ADAMS	1100	12-JAN-1983	N/A
30	ALLEN	1600	20-FEB-1981	350
30	WARD	1250	22-FEB-1981	-1600
30	BLAKE	2850	01-MAY-1981	1350
30	TURNER	1500	08-SEP-1981	250
30	MARTIN	1250	28-SEP-1981	300
30	JAMES	950	03-DEC-1981	N/A

## 解决方案

这是使用 Oracle 视窗函数 LEAD OVER 和 LAG OVER 的另一个例子。这样可以很容易地访问下一行及前一行，而无需进行额外联接。对于其他 RDBMS，可以使用标量子查询，但比这要复杂。如果只能使用标量子查询或自联接方式来解决的话，这个问题本身就不怎么好。

## DB2、MySQL、PostgreSQL 和 SQL Server

使用标量子查询，检索紧随各员工之后聘用的员工的 HIREDATE。然后，使用另一个标量子查询，查找这个（后聘用）员工的工资：

```

1 select deptno,ename,hiredate,sal,
2      coalesce(cast(sal-next_sal as char(10)),'N/A') as diff
3   from (
4   select e.deptno,
5          e.ename,
6          e.hiredate,
7          e.sal,
8          (select min(sal) from emp d
9           where d.deptno=e.deptno
10            and d.hiredate =
11              (select min(hiredate) from emp d
12               where e.deptno=d.deptno
13                and d.hiredate > e.hiredate)) as next_sal
14   from emp e
15  ) x

```

## Oracle

使用视窗函数 LEAD OVER，访问相对于当前行的“下一个”员工的工资：

```

1 select deptno,ename,sal,hiredate,
2      lpad(nvl(to_char(sal-next_sal),'N/A'),10) diff
3   from (
4   select deptno,ename,sal,hiredate,
5          lead(sal)over(partition by deptno
6                        order by hiredate) next_sal
7   from emp
8  )

```

## 讨论

## DB2、MySQL、PostgreSQL 和 SQL Server

使用标量子查询，检索同一部门中紧随各员工之后聘用的员工的 HIREDATE。在标量子查询中，该解决方案使用了 MIN(HIREDATE)，以确保返回唯一值，即使在同一天聘用很多人也是如此：

```

select e.deptno,
       e.ename,
       e.hiredate,
       e.sal,
       (select min(hiredate) from emp d
        where e.deptno=d.deptno
         and d.hiredate > e.hiredate) as next_hire
from emp e
order by 1

```

DEPTNO	ENAME	HIREDATE	SAL	NEXT_HIRE
10	CLARK	09-JUN-1981	2450	17-NOV-1981
10	KING	17-NOV-1981	5000	23-JAN-1982
10	MILLER	23-JAN-1982	1300	
20	SMITH	17-DEC-1980	800	02-APR-1981
20	ADAMS	12-JAN-1983	1100	
20	FORD	03-DEC-1981	3000	09-DEC-1982
20	SCOTT	09-DEC-1982	3000	12-JAN-1983
20	JONES	02-APR-1981	2975	03-DEC-1981
30	ALLEN	20-FEB-1981	1600	22-FEB-1981
30	BLAKE	01-MAY-1981	2850	08-SEP-1981
30	MARTIN	28-SEP-1981	1250	03-DEC-1981
30	JAMES	03-DEC-1981	950	
30	TURNER	08-SEP-1981	1500	28-SEP-1981
30	WARD	22-FEB-1981	1250	01-MAY-1981

接下来, 使用另一个标量子查询, 查找NEXT\_HIRE日期聘用的员工的工资。这里也使用了MIN函数, 以确保总是返回唯一值:

```

select e.deptno,
       e.ename,
       e.hiredate,
       e.sal,
       (select min(sal) from emp d
        where d.deptno=e.deptno
         and d.hiredate =
              (select min(hiredate) from emp d
               where e.deptno=d.deptno
                and d.hiredate > e.hiredate)) as next_sal
from emp e
order by 1

```

DEPTNO	ENAME	HIREDATE	SAL	NEXT_SAL
10	CLARK	09-JUN-1981	2450	5000
10	KING	17-NOV-1981	5000	1300
10	MILLER	23-JAN-1982	1300	
20	SMITH	17-DEC-1980	800	2975
20	ADAMS	12-JAN-1983	1100	
20	FORD	03-DEC-1981	3000	3000
20	SCOTT	09-DEC-1982	3000	1100
20	JONES	02-APR-1981	2975	3000
30	ALLEN	20-FEB-1981	1600	1250
30	BLAKE	01-MAY-1981	2850	1500
30	MARTIN	28-SEP-1981	1250	950
30	JAMES	03-DEC-1981	950	
30	TURNER	08-SEP-1981	1500	1250
30	WARD	22-FEB-1981	1250	2850

最后一步, 求SAL和NEXT\_SAL之差, 必要的情况下, 可使用函数COALESCE返回“N/A”。由于差值是一个数字, 而且有可能是NULL值, 那么必须把差值转换为字符串, 传给COALESCE:

```

select deptno,ename,hiredate,sal,
       coalesce(cast(sal-next_sal as char(10)), 'N/A') as diff
  from (
select e.deptno,
       e.ename,
       e.hiredate,
       e.sal,
       (select min(sal) from emp d
        where d.deptno=e.deptno
        and d.hiredate =
              (select min(hiredate) from emp d
               where e.deptno=d.deptno
               and d.hiredate > e.hiredate)) as next_sal
  from emp e
  ) x
 order by 1

```

DEPTNO	ENAME	HIREDATE	SAL	DIFF
10	CLARK	09-JUN-1981	2450	-2550
10	KING	17-NOV-1981	5000	3700
10	MILLER	23-JAN-1982	1300	N/A
20	SMITH	17-DEC-1980	800	-2175
20	ADAMS	12-JAN-1983	1100	N/A
20	FORD	03-DEC-1981	3000	0
20	SCOTT	09-DEC-1982	3000	1900
20	JONES	02-APR-1981	2975	-25
30	ALLEN	20-FEB-1981	1600	350
30	BLAKE	01-MAY-1981	2850	1350
30	MARTIN	28-SEP-1981	1250	300
30	JAMES	03-DEC-1981	950	N/A
30	TURNER	08-SEP-1981	1500	250
30	WARD	22-FEB-1981	1250	-1600

注意：这解决方案中使用 MIN(SAL) 是一个例子，说明如何无意间以某种方式把业务逻辑放到查询中，而看起来似乎只是纯技术性的决定。如果对于一个给定日期有很多工资数字，应该取最低的、最高的还是平均值呢？本例选择了最低工资值，而在现实生活中，可能会让要求报表的业务客户来进行决策。

## Oracle

第一步，使用 LEAD OVER 视窗函数，为每个员工找到同一部门内的“下一个”工资。对于每个部门中最新聘用的员工，他的 NEXT\_SAL 值为 NULL：

```

select deptno,ename,sal,hiredate,
       lead(sal)over(partition by deptno order by hiredate) next_sal
  from emp

```

DEPTNO	ENAME	SAL	HIREDATE	NEXT_SAL
10	CLARK	2450	09-JUN-1981	5000
10	KING	5000	17-NOV-1981	1300
10	MILLER	1300	23-JAN-1982	
20	SMITH	800	17-DEC-1980	2975
20	JONES	2975	02-APR-1981	3000
20	FORD	3000	03-DEC-1981	3000
20	SCOTT	3000	09-DEC-1982	1100
20	ADAMS	1100	12-JAN-1983	
30	ALLEN	1600	20-FEB-1981	1250
30	WARD	1250	22-FEB-1981	2850
30	BLAKE	2850	01-MAY-1981	1500

30	TURNER	1500	08-SEP-1981	1250
30	MARTIN	1250	28-SEP-1981	950
30	JAMES	950	03-DEC-1981	

下一步，获取每个员工与同一部门中紧随其后聘用的员工的工资之差：

```
select deptno,ename,sal,hiredate, sal-next_sal diff
  from (
select deptno,ename,sal,hiredate,
       lead(sal)over(partition by deptno order by hiredate) next_sal
  from emp
)
```

DEPTNO	ENAME	SAL	HIREDATE	DIFF
10	CLARK	2450	09-JUN-1981	-2550
10	KING	5000	17-NOV-1981	3700
10	MILLER	1300	23-JAN-1982	
20	SMITH	800	17-DEC-1980	-2175
20	JONES	2975	02-APR-1981	-25
20	FORD	3000	03-DEC-1981	0
20	SCOTT	3000	09-DEC-1982	1900
20	ADAMS	1100	12-JAN-1983	
30	ALLEN	1600	20-FEB-1981	350
30	WARD	1250	22-FEB-1981	-1600
30	BLAKE	2850	01-MAY-1981	1350
30	TURNER	1500	08-SEP-1981	250
30	MARTIN	1250	28-SEP-1981	300
30	JAMES	950	03-DEC-1981	

当 DIFF 为 NULL 时，可使用函数 NVL 返回 “N/A”。要返回 “N/A”，必须先把 DIFF 值转换为字符串，否则 NVL 不能正常执行：

```
select deptno,ename,sal,hiredate,
       nvl(to_char(sal-next_sal),'N/A') diff
  from (
select deptno,ename,sal,hiredate,
       lead(sal)over(partition by deptno order by hiredate) next_sal
  from emp
)
```

DEPTNO	ENAME	SAL	HIREDATE	DIFF
10	CLARK	2450	09-JUN-1981	-2550
10	KING	5000	17-NOV-1981	3700
10	MILLER	1300	23-JAN-1982	N/A
20	SMITH	800	17-DEC-1980	-2175
20	JONES	2975	02-APR-1981	-25
20	FORD	3000	03-DEC-1981	0
20	SCOTT	3000	09-DEC-1982	1900
20	ADAMS	1100	12-JAN-1983	N/A
30	ALLEN	1600	20-FEB-1981	350
30	WARD	1250	22-FEB-1981	-1600
30	BLAKE	2850	01-MAY-1981	1350
30	TURNER	1500	08-SEP-1981	250
30	MARTIN	1250	28-SEP-1981	300
30	JAMES	950	03-DEC-1981	N/A

最后一步，使用函数 LPAD 设置 DIFF 值的格式。默认情况下，数字都是右对齐，而字符串都是左对齐。使用 LPAD，可使这个列中的所有结果右对齐：

```
select deptno,ename,sal,hiredate,
       lpad(nvl(to_char(sal-next_sal),'N/A'),10) diff
  from (
```



```
select deptno,ename,sal,hiredate,
       lead(sal)over(partition by deptno order by hiredate) next_sal
from emp
)
```

DEPTNO	ENAME	SAL	HIREDATE	DIFF
10	CLARK	2450	09-JUN-1981	-2550
10	KING	5000	17-NOV-1981	3700
10	MILLER	1300	23-JAN-1982	N/A
20	SMITH	800	17-DEC-1980	-2175
20	JONES	2975	02-APR-1981	-25
20	FORD	3000	03-DEC-1981	0
20	SCOTT	3000	09-DEC-1982	1900
20	ADAMS	1100	12-JAN-1983	N/A
30	ALLEN	1600	20-FEB-1981	350
30	WARD	1250	22-FEB-1981	-1600
30	BLAKE	2850	01-MAY-1981	1350
30	TURNER	1500	08-SEP-1981	250
30	MARTIN	1250	28-SEP-1981	300
30	JAMES	950	03-DEC-1981	N/A

本书中讲述的大部分解决方案都没有处理“假设”的场景（这是作者考虑再三的结果，而且这样可以使本书的可读性更好），当使用 Oracle 的 LEAD OVER 函数时，会存在重复现象，这种情况则一定要讨论。对于表 EMP 中的简单样本数据，并不存在重复的 HIREDATE 值，然而，这只是一种可能的情形。通常情况下，本书不会讨论“假设”场景，如重复数据（因为表 EMP 中不存在重复），但是涉及 LEAD 的应对措施（尤其是对没有 Oracle 背景的读者）可能不是这么明显。下面的查询将返回 DEPTNO 10 中员工之间 SAL 的差值（差值是按员工的聘用顺序计算的）：

```
select deptno,ename,sal,hiredate,
       lpad(nvl(to_char(sal-next_sal),'N/A'),10) diff
from (
select deptno,ename,sal,hiredate,
       lead(sal)over(partition by deptno
                     order by hiredate) next_sal
from emp
where deptno=10 and empno > 10
)
```

DEPTNO	ENAME	SAL	HIREDATE	DIFF
10	CLARK	2450	09-JUN-1981	-2550
10	KING	5000	17-NOV-1981	3700
10	MILLER	1300	23-JAN-1982	N/A

基于表 EMP 中的数据，该解决方案是正确的；但是，如果存在重复的行，这个解决方案就会失败。请看下列例子，在聘用 KING 的同一天，还聘用了 4 个员工：

```
insert into emp (empno,ename,deptno,sal,hiredate)
values (1,'ant',10,1000,to_date('17-NOV-1981'))

insert into emp (empno,ename,deptno,sal,hiredate)
values (2,'joe',10,1500,to_date('17-NOV-1981'))

insert into emp (empno,ename,deptno,sal,hiredate)
values (3,'jim',10,1600,to_date('17-NOV-1981'))

insert into emp (empno,ename,deptno,sal,hiredate)
values (4,'jon',10,1700,to_date('17-NOV-1981'))

select deptno,ename,sal,hiredate,
```

```

        lpad(nvl(to_char(sal-next_sal),'N/A'),10) diff
    from (
select deptno,ename,sal,hiredate,
        lead(sal)over(partition by deptno
                      order by hiredate) next_sal
    from emp
   where deptno=10
    )

```

DEPTNO	ENAME	SAL	HIREDATE	DIFF
10	CLARK	2450	09-JUN-1981	1450
10	ant	1000	17-NOV-1981	-500
10	joe	1500	17-NOV-1981	-3500
10	KING	5000	17-NOV-1981	3400
10	jim	1600	17-NOV-1981	-100
10	jon	1700	17-NOV-1981	400
10	MILLER	1300	23-JAN-1982	N/A

注意，除员工 JON 之外，同一天（11 月 17 日）聘用的所有员工都是相对于同一天聘用的其他员工计算工资差的，这种做法是错误的。11 月 17 日聘用的所有员工都应该相对于 MILLER 的工资计算工资差值，而不是针对 17 日聘用的其他员工进行计算。以员工 ANT 为例，ANT 的 DIFF 值 -500，这是由于在拿 ANT 的 SAL 与 JOE 的 SAL 相比较，他的 SAL 比 JOE 的 SAL 少 500，因此等于 -500；而 ANT 正确的 DIFF 值应该是 -300，因为 ANT 的 SAL 比 MILLER 的 SAL 少 300，而 HIREDATE MILLER 是后一个聘用的员工。该解决方案不正确的原因是 Oracle 中 LEAD OVER 函数的默认行为，默认情况下，LEAD OVER 只会向前查一行，因此，对于员工 ANT 而言，按照 HIREDATE 的下一个 SAL 是 JOE 的 SAL，因为 LEAD OVER 只会向前查看一行，而不会跳过重复行。幸运的是，Oracle 考虑到了这一点，它允许给 LEAD OVER 传递一个参数，以便确定应该向前检查多少行。对上述例子，正确的解决方案只要再数一数：数出 11 月 17 日聘用的各员工距离 1 月 23 日（MILLER 的 HIREDATE）多少行。下面列出的解决方案实现了这种功能：

```

select deptno,ename,sal,hiredate,
        lpad(nvl(to_char(sal-next_sal),'N/A'),10) diff
    from (
select deptno,ename,sal,hiredate,
        lead(sal,cnt-rn+1)over(partition by deptno
                              order by hiredate) next_sal
    from (
select deptno,ename,sal,hiredate,
        count(*)over(partition by deptno,hiredate) cnt,
        row_number()over(partition by deptno,hiredate order by sal) rn
    from emp
   where deptno=10
    )
    )

```

DEPTNO	ENAME	SAL	HIREDATE	DIFF
10	CLARK	2450	09-JUN-1981	1450
10	ant	1000	17-NOV-1981	-300
10	joe	1500	17-NOV-1981	200
10	jim	1600	17-NOV-1981	300
10	jon	1700	17-NOV-1981	400
10	KING	5000	17-NOV-1981	3700
10	MILLER	1300	23-JAN-1982	N/A

现在，该解决方案是正确的，可以看到，11 月 17 日聘用的所有员工的工资都是在与

MILLER 的工资相比较。检验一下结果，员工 ANT 的 DIFF 值为 -300，这正是我们希望得到的值。如果对传递给 LEAD OVER 的表达式还不够清楚，CNT-RN+1 就表示 11 月 17 日聘用的各员工距 MILLER 的行数。下面的内联视图显示了 CNT 和 RN 的值：

```
select deptno,ename,sal,hiredate,
       count(*)over(partition by deptno,hiredate) cnt,
       row_number()over(partition by deptno,hiredate order by sal) rn
from emp
where deptno=10
```

DEPTNO	ENAME	SAL	HIREDATE	CNT	RN
10	CLARK	2450	09-JUN-1981	1	1
10	ant	1000	17-NOV-1981	5	1
10	joe	1500	17-NOV-1981	5	2
10	jim	1600	17-NOV-1981	5	3
10	jon	1700	17-NOV-1981	5	4
10	KING	5000	17-NOV-1981	5	5
10	MILLER	1300	23-JAN-1982	1	1

对于包含重复 HIREDATE 值的所有员工，CNT 值表示 HIREDATE 的重复次数，RN 值表示 DEPTNO 10 中员工的序号，该序号是按 DEPTNO 和 HIREDATE 分区，这样，只有那些与其他员工具有相同 HIREDATE 的员工，这个值才可能大于 1。该序号按 SAL 的顺序（这只是随意选择的，SAL 使用起来很方便，选择 EMPNO 也一样容易）分配。现在，知道了总的重复个数，而且每个重复值都有一个序号，与 MILLER 之间的距离就是重复数减去当前序号值再加 1（即 CNT-RN+1）。有关距离计算的结果及其对 LEAD OVER 的影响，如下所示：

```
select deptno,ename,sal,hiredate,
       lead(sal)over(partition by deptno
                     order by hiredate) incorrect,
       cnt-rn+1 distance,
       lead(sal,cnt-rn+1)over(partition by deptno
                              order by hiredate) correct
from (
select deptno,ename,sal,hiredate,
       count(*)over(partition by deptno,hiredate) cnt,
       row_number()over(partition by deptno,hiredate
                        order by sal) rn
from emp
where deptno=10
)
```

DEPTNO	ENAME	SAL	HIREDATE	INCORRECT	DISTANCE	CORRECT
10	CLARK	2450	09-JUN-1981	1000	1	1000
10	ant	1000	17-NOV-1981	1500	5	1300
10	joe	1500	17-NOV-1981	1600	4	1300
10	jim	1600	17-NOV-1981	1700	3	1300
10	jon	1700	17-NOV-1981	5000	2	1300
10	KING	5000	17-NOV-1981	1300	1	1300
10	MILLER	1300	23-JAN-1982		1	

现在，可以清楚地看到给 LEAD OVER 传递正确距离值后的效果。INCORRECT 行表示使用默认距离时由 LEAD OVER 返回的值，CORRECT 表示使用正确距离（那些 HIREDATE 重复的员工与 MILLER 相差的行数）时由 LEAD OVER 返回的值。接下来，就可以返回的每一行查找 CORRECT 和 SAL 之差，这前面已经看到了。

## 10.3 定位连续值范围的开始点和结束点问题

本节是上一节的扩展，它也使用了上一节的视图 V。得到连续值的范围后，就可以查找它们的开始点和结束点。与上一节不同的是，如果某行不属于一组连续值，仍然要返回它。为什么呢？因为这样的行既表示了它的范围的开始点，又表示结束点。下面的例子使用了视图 V 中的数据：

```
select *
  from V
PROJ_ID PROJ_START PROJ_END
-----
1 01-JAN-2005 02-JAN-2005
2 02-JAN-2005 03-JAN-2005
3 03-JAN-2005 04-JAN-2005
4 04-JAN-2005 05-JAN-2005
5 06-JAN-2005 07-JAN-2005
6 16-JAN-2005 17-JAN-2005
7 17-JAN-2005 18-JAN-2005
8 18-JAN-2005 19-JAN-2005
9 19-JAN-2005 20-JAN-2005
10 21-JAN-2005 22-JAN-2005
11 26-JAN-2005 27-JAN-2005
12 27-JAN-2005 28-JAN-2005
13 28-JAN-2005 29-JAN-2005
14 29-JAN-2005 30-JAN-2005
```

最终结果集如下：

```
PROJ_GRP PROJ_START PROJ_END
-----
1 01-JAN-2005 05-JAN-2005
2 06-JAN-2005 07-JAN-2005
3 16-JAN-2005 20-JAN-2005
4 21-JAN-2005 22-JAN-2005
5 26-JAN-2005 30-JAN-2005
```

### 解决方案

这个问题比前一个问题更为复杂。首先，必须识别它们的范围。PROJ\_START 和 PROJ\_END 的值定义了一行的范围，要把某行看作“连续的”或者属于一个组，则它的 PROJ\_START 值一定要等于它前一行的 PROJ\_END 值；如果某个行的 PROJ\_START 值不等于前一行的 PROJ\_END 值，而且它的 PROJ\_END 值也不等于下一行的 PROJ\_START 值，则它就是一个单个行组。确定了范围之后，就需要将属于同一范围的行划成同一组，并且只返回组开始点和结束点。

检验目标结果集的第一行。PROJ\_START 是视图 V 中 PROJ\_ID 1 的 PROJ\_START，而且 PROJ\_END 是视图 V 中 PROJ\_ID 4 的 PROJ\_END。尽管 PROJ\_ID 4 之后并没有连续工程，但它是连续值范围的最后一个值，因此把它编入第一组。

## DB2、MySQL、PostgreSQL 和 SQL Server

对于这些平台，其解决方案要使用视图 V2，以提高可读性。视图 V2 的定义如下：

```
create view v2
as
select a.*,
       case
         when (
               select b.proj_id
               from V b
               where a.proj_start = b.proj_end
             )
         is not null then 0 else 1
       end as flag
from V a
```

视图 V2 的结果集如下：

```
select *
from V2
```

PROJ_ID	PROJ_START	PROJ_END	FLAG
1	01-JAN-2005	02-JAN-2005	1
2	02-JAN-2005	03-JAN-2005	0
3	03-JAN-2005	04-JAN-2005	0
4	04-JAN-2005	05-JAN-2005	0
5	06-JAN-2005	07-JAN-2005	1
6	16-JAN-2005	17-JAN-2005	1
7	17-JAN-2005	18-JAN-2005	0
8	18-JAN-2005	19-JAN-2005	0
9	19-JAN-2005	20-JAN-2005	0
10	21-JAN-2005	22-JAN-2005	1
11	26-JAN-2005	27-JAN-2005	1
12	27-JAN-2005	28-JAN-2005	0
13	28-JAN-2005	29-JAN-2005	0
14	29-JAN-2005	30-JAN-2005	0

下面给出了使用 V2 的解决方案。首先，找到那些属于一组连续值的行。给这些行分组，然后使用 MIN 和 MAX 函数，找到它们的开始点和结束点：

```
1 select proj_grp,
2        min(proj_start) as proj_start,
3        max(proj_end) as proj_end
4   from (
5   select a.proj_id,a.proj_start,a.proj_end,
6          (select sum(b.flag)
7           from V2 b
8           where b.proj_id <= a.proj_id) as proj_grp
9   from V2 a
10  ) x
11  group by proj_grp
```

## Oracle

尽管其他平台的解决方案适用于 Oracle，但是利用 Oracle 的 LAG OVER 窗口函数就不需要引入额外的视图。使用 LAG OVER，确定前一行的 PROJ\_END 是否等于当前行的 PROJ\_START，这样可判断是否把行放入组中。给它们分组之后，就可以使用聚集函数 MIN 和 MAX 找到它们的开始点和结束点：

```

1 select proj_grp, min(proj_start), max(proj_end)
2   from (
3 select proj_id,proj_start,proj_end,
4        sum(flag)over(order by proj_id) proj_grp
5   from (
6 select proj_id,proj_start,proj_end,
7        case when
8            lag(proj_end)over(order by proj_id) = proj_start
9            then 0 else 1
10       end flag
11   from v
12  )
13  )
14 group by proj_grp

```

## 讨论

### DB2、MySQL、PostgreSQL 和 SQL Server

采用视图 V2 可以很容易地解决这个问题。在视图 V2 的 CASE 表达式中, 使用了标量子查询, 它用于确定某个特殊行是否属于一组连续值。如果当前行属于连续集, 则 CASE 表达式 (别名 FLAG) 将返回 0, 否则将返回 1 (判断是否属于连续行集的依据是: 是否存在某个记录的 PROJ\_END 值是否与当前行的 PROJ\_START 值相匹配)。下一步, 检验内联视图 X (第 5~9 行)。内联视图 X 返回视图 V2 中的所有行及 FLAG 的累加和, 这个累加和用于创建组, 如下所示:

```

select a.proj_id,a.proj_start,a.proj_end,
       (select sum(b.flag)
        from v2 b
        where b.proj_id <= a.proj_id) as proj_grp
  from v2 a

```

PROJ_ID	PROJ_START	PROJ_END	PROJ_GRP
1	01-JAN-2005	02-JAN-2005	1
2	02-JAN-2005	03-JAN-2005	1
3	03-JAN-2005	04-JAN-2005	1
4	04-JAN-2005	05-JAN-2005	1
5	06-JAN-2005	07-JAN-2005	2
6	16-JAN-2005	17-JAN-2005	3
7	17-JAN-2005	18-JAN-2005	3
8	18-JAN-2005	19-JAN-2005	3
9	19-JAN-2005	20-JAN-2005	3
10	21-JAN-2005	22-JAN-2005	4
11	26-JAN-2005	27-JAN-2005	5
12	27-JAN-2005	28-JAN-2005	5
13	28-JAN-2005	29-JAN-2005	5
14	29-JAN-2005	30-JAN-2005	5

至此, 已经给范围分了组, 下面可以分别将聚集函数 MIN 和 MAX 用于 PROJ\_START 和 PROJ\_END 列, 并累加和的值分组, 以求出每组的开始点和结束点。

### Oracle

在这种情形下窗口函数 LAG OVER 特别有用。可以检验前一行的 PROJ\_END 值, 而不需要自联接、标量子查询或其他视图。没有使用 CASE 表达式的 LAG OVER 函数的结果如下所示:



```
select proj_id,proj_start,proj_end,
       lag(proj_end)over(order by proj_id) prior_proj_end
from v
```

PROJ_ID	PROJ_START	PROJ_END	PRIOR_PROJ_END
1	01-JAN-2005	02-JAN-2005	
2	02-JAN-2005	03-JAN-2005	02-JAN-2005
3	03-JAN-2005	04-JAN-2005	03-JAN-2005
4	04-JAN-2005	05-JAN-2005	04-JAN-2005
5	06-JAN-2005	07-JAN-2005	05-JAN-2005
6	16-JAN-2005	17-JAN-2005	07-JAN-2005
7	17-JAN-2005	18-JAN-2005	17-JAN-2005
8	18-JAN-2005	19-JAN-2005	18-JAN-2005
9	19-JAN-2005	20-JAN-2005	19-JAN-2005
10	21-JAN-2005	22-JAN-2005	20-JAN-2005
11	26-JAN-2005	27-JAN-2005	22-JAN-2005
12	27-JAN-2005	28-JAN-2005	27-JAN-2005
13	28-JAN-2005	29-JAN-2005	28-JAN-2005
14	29-JAN-2005	30-JAN-2005	29-JAN-2005

完整解决方案中用 CASE 表达式将 LAG OVER 返回的值与当前行的 PROJ\_START 值相比较；如果它们相同，则返回 0，否则返回 1。然后，针对 CASE 表达式返回的 0 或 1 值创建累加和，以便把每一行都放入组中。累加和的结果如下所示：

```
select proj_id,proj_start,proj_end,
       sum(flag)over(order by proj_id) proj_grp
from (
select proj_id,proj_start,proj_end,
       case when
         lag(proj_end)over(order by proj_id) = proj_start
         then 0 else 1
       end flag
from v
)
```

PROJ_ID	PROJ_START	PROJ_END	PROJ_GRP
1	01-JAN-2005	02-JAN-2005	1
2	02-JAN-2005	03-JAN-2005	1
3	03-JAN-2005	04-JAN-2005	1
4	04-JAN-2005	05-JAN-2005	1
5	06-JAN-2005	07-JAN-2005	2
6	16-JAN-2005	17-JAN-2005	3
7	17-JAN-2005	18-JAN-2005	3
8	18-JAN-2005	19-JAN-2005	3
9	19-JAN-2005	20-JAN-2005	3
10	21-JAN-2005	22-JAN-2005	4
11	26-JAN-2005	27-JAN-2005	5
12	27-JAN-2005	28-JAN-2005	5
13	28-JAN-2005	29-JAN-2005	5
14	29-JAN-2005	30-JAN-2005	5

至此，已经把每一行都放入组中，下面可以将聚集函数 MIN 和 MAX 分别用于 PROJ\_START 和 PROJ\_END 列，并按 PROJ\_GRP 累加和列进行分组。

## 10.4 补充范围内丢失的值

### 问题

返回 1980 年起始的十年间每年聘用的员工数，但有些年份并没有聘用员工。应该返回下列结果集：

YR	CNT
1980	1
1981	10
1982	2
1983	1
1984	0
1985	0
1986	0
1987	0
1988	0
1989	0

## 解决方案

这个解决方案的技巧是，对未聘用任何员工的年份返回0。如果某一给定年份没有聘用员工，那么表EMP中就不存在这一年的有关记录。如果表中不包含某个年份，那么如何才能返回一个计数值、任意数抑或0呢？这种解决方案需要进行外联接，必须提供一个结果集，它包含所有想要查看的年份，然后对表EMP进行计数，以便查看这些年内是否聘用了员工。

### DB2

把表EMP当作基干表使用（因为它包含14行），而且使用内置函数YEAR，为1980起始的十年间的每一年生成一行信息。对表EMP进行外联接，并计数每年聘用的员工数：

```

1 select x.yr, coalesce(y.cnt,0) cnt
2   from (
3 select year(min(hiredate)over()) -
4         mod(year(min(hiredate)over()),10) +
5         row_number()over()-1 yr
6   from emp fetch first 10 rows only
7   ) x
8  left join
9   (
10 select year(hiredate) yr1, count(*) cnt
11   from emp
12  group by year(hiredate)
13   ) y
14   on ( x.yr = y.yr1 )

```

### Oracle

把表EMP当作基干表使用（因为它包含14行），而且使用内置函数TO\_NUMBER和TO\_CHAR，为1980起始的十年间的每一年生成一行信息。对表EMP进行外联接，并计数每年聘用的员工数：

```

1 select x.yr, coalesce(cnt,0) cnt
2   from (
3 select extract(year from min(hiredate)over()) -
4         mod(extract(year from min(hiredate)over()),10) +
5         rownum-1 yr
6   from emp
7  where rownum <= 10
8   ) x,
9   (
10 select to_number(to_char(hiredate,'YYYY')) yr, count(*) cnt
11   from emp

```



```

12 group by to_number(to_char(hiredate, 'YYYY'))
13 ) y
14 where x.yr = y.yr(+)

```

在 Oracle9i Database 或更高版本中，可以使用最新的 JOIN 子句实现这种解决方案：

```

1 select x.yr, coalesce(cnt,0) cnt
2   from (
3 select extract(year from min(hiredate)over()) -
4        mod(extract(year from min(hiredate)over()),10) +
5        rownum-1 yr
6   from emp
7  where rownum <= 10
8   ) x
9  left join
10 (
11 select to_number(to_char(hiredate, 'YYYY')) yr, count(*) cnt
12   from emp
13  group by to_number(to_char(hiredate, 'YYYY'))
14   ) y
15   on ( x.yr = y.yr )

```

## PostgreSQL 和 MySQL

把表 T10 当作基干表使用（因为它包含 10 行），并使用内置函数 EXTRACT，为 1980 起始的十年间的每一年生成一行信息。对表 EMP 进行外联接，并计数每年聘用的员工数：

```

1 select y.yr, coalesce(x.cnt,0) as cnt
2   from (
3 select min_year-mod(cast(min_year as int),10)+rn as yr
4   from (
5 select (select min(extract(year from hiredate))
6        from emp) as min_year,
7        id-1 as rn
8   from t10
9   ) a
10   ) y
11  left join
12 (
13 select extract(year from hiredate) as yr, count(*) as cnt
14   from emp
15  group by extract(year from hiredate)
16   ) x
17   on ( y.yr = x.yr )

```

## SQL Server

把表 EMP 当作基干表使用（因为它包含 14 行），而且使用内置函数 YEAR，为 1980 起始的十年间的每一年生成一行信息。对表 EMP 进行外联接，并计数每年聘用的员工数：

```

1 select x.yr, coalesce(y.cnt,0) cnt
2   from (
3 select top (10)
4        (year(min(hiredate)over()) -
5         year(min(hiredate)over())%10)+
6         row_number()over(order by hiredate)-1 yr
7   from emp
8   ) x
9  left join
10 (
11 select year(hiredate) yr, count(*) cnt
12   from emp
13  group by year(hiredate)

```

```

14         ) y
15     on ( x.yr = y.yr )

```

## 讨论

对于上述这些解决方案，尽管它们的语法不同，但步骤是相同的。内联视图 X 先找到最早的 HIREDATE 年份，从而返回 1980 起始的十年间的所有年份；接下来，计算最早年份与它对 10 的模之间的差，再给这个差加 RN-1。要了解其机理，只需执行内联视图 X，并分别返回每个相关值。以下分别给出了使用窗口函数 MIN OVER (DB2, Oracle, SQL Server) 和标量子查询(MySQL, PostgreSQL)的内联视图 X 及其返回的结果集：

```

select year(min(hiredate)over()) -
       mod(year(min(hiredate)over()),10) +
       row_number()over()-1 yr,
       year(min(hiredate)over()) min_year,
       mod(year(min(hiredate)over()),10) mod_yr,
       row_number()over()-1 rn
from emp fetch first 10 rows only

```

YR	MIN_YEAR	MOD_YR	RN
1980	1980	0	0
1981	1980	0	1
1982	1980	0	2
1983	1980	0	3
1984	1980	0	4
1985	1980	0	5
1986	1980	0	6
1987	1980	0	7
1988	1980	0	8
1989	1980	0	9

```

select min_year-mod(min_year,10)+rn as yr,
       min_year,
       mod(min_year,10) as mod_yr
       rn
from (
select (select min(extract(year from hiredate))
       from emp) as min_year,
       id-1 as rn
from t10
) x

```

YR	MIN_YEAR	MOD_YR	RN
1980	1980	0	0
1981	1980	0	1
1982	1980	0	2
1983	1980	0	3
1984	1980	0	4
1985	1980	0	5
1986	1980	0	6
1987	1980	0	7
1988	1980	0	8
1989	1980	0	9

内联视图 Y 返回了每个 HIREDATE 的年份以及这一年聘用的员工数：

```

select year(hiredate) yr, count(*) cnt
from emp
group by year(hiredate)

```

YR	CNT
1980	1
1981	10
1982	2
1983	1

最终解决方案就是把内联视图 Y 与内联视图 X 进行外部联接，这样就会返回所有年份，而不管那一年是否聘用了员工。

## 10.5 生成连续数字值

### 问题

在查询中使用一个“行源生成器”。对于需要行骨架的查询来说，行源生成器是非常有用的。例如，按指定的行数返回以下结果集：

```
ID
---
1
2
3
4
5
6
7
8
9
10
...
```

如果 RDBMS 提供了能够动态返回行的内置函数，就不需要提前创建包含固定行数的基干表。这就是动态行生成器好用的原因。否则，需要时，必须使用包含固定行数的传统基干表（这不可能总够用）生成行。

### 解决方案

这种解决方案说明了如何返回从 1 开始计数的 10 行信息。可以很容易地对这种解决方案加以修改，以返回任意数目的行。

如果能够返回值从 1 开始递增的连续值，就为其他许多解决方案打开了大门。例如，可以把生成的数字与日期相加，以便生成日期序列；也可以使用这些数解析字符串。

### DB2 和 SQL Server

使用递归 WITH 子句生成带有递增值的行序列。使用一行的表（如 T1）开始生成行，其余的工作由 WITH 子句完成：

```
1 with x (id)
2 as (
3   select 1
4     from t1
5   union all
6   select id+1
```

```

7   from x
8   where id+1 <= 10
9   )
10  select * from x

```

下面给出了第二种解决方案，这种方案只适用于 DB2。它的优点是不需要表 T1：

```

1  with x (id)
2  as (
3  values '(1)
4    union all
5  select id+1
6    from x
7    where id+1 <= 10
8  )
9  select * from x

```

## Oracle

使用递归 CONNECT BY 子句 (Oracle9i Database 或更高版本)。在 Oracle 9i Database 中，要么把 CONNECT BY 放在内联视图中，要么把它放在 WITH 子句中：

```

1  with x
2  as (
3  select level id
4    from dual
5    connect by level <= 10
6  )
7  select * from x

```

在 Oracle Database 10g 或更高版本中，可以使用 MODEL 子句生成行：

```

1  select array id
2    from dual
3    model
4      dimension by (0 idx)
5      measures(1 array)
6      rules iterate (10) (
7        array[iteration_number] = iteration_number+1
8      )

```

## PostgreSQL

使用函数 GENERATE\_SERIES，它是为生成行而专门设计的：

```

1  select id
2    from generate_series (1,10) x(id)

```

## 讨论

### DB2 和 SQL Server

递归 WITH 子句将递增 ID (从 1 开始) 值，直到满足 WHERE 子句的条件为止。为启动行生成，必须生成包含值 1 的一行信息，可以对一行的表用 SELECT 1，对于 DB2 系统，= 可以使用 VALUES 子句创建一行的结果集。

## Oracle

该解决方案是把 CONNECT BY 子查询放入 WITH 子句中。在未满足 WHERE 子句的条

件之前，会一直返回行。Oracle 会自动递增伪列（pseudo-column）LEVEL 值，因此不需要手动做这件事。

在 MODEL 子句的解决方案中，使用了显式的 ITERATE 命令，以生成多个行。如果没有 ITERATE 子句，只会返回一行，因为 DUAL 中只有一行。例如：

```
select array id
  from dual
 model
   dimension by (0 idx)
   measures(1 array)
   rules ()

ID
--
1
```

使用 MODEL 子句不仅可以像数组一样访问行，而且可以很容易地“创建”或返回 SELECT 的对象表中所没有的行。在这种解决方案中，IDX 是数组下标（数组中特定值的位置），而且 ARRAY（别名 ID）是行的“数组”。第一行默认为 1，可以使用 ARRAY[0] 进行引用。Oracle 提供了函数 ITERATION\_NUMBER，可以跟踪迭代次数。该解决方案迭代执行了 10 次，因此 ITERATION\_NUMBER 的值为 0~9。分别给这些值加 1，就能得到 1~10 的结果。

如果执行下列查询，可能更容易想象采 model 子句的工作过程：

```
select 'array['||idx||'] = '||array as output
  from dual
 model
   dimension by (0 idx)
   measures(1 array)
   rules iterate (10) (
     array[iteration_number] = iteration_number+1
   )

OUTPUT
-----
array[0] = 1
array[1] = 2
array[2] = 3
array[3] = 4
array[4] = 5
array[5] = 6
array[6] = 7
array[7] = 8
array[8] = 9
array[9] = 10
```

## PostgreSQL

所有工作都是由函数 GENERATE\_SERIES 完成的。该函数接收 3 个参数，它们都是数值型的。第一个参数是开始值，第二个参数是结束值，第三个参数是“步长”值（每个值增加多少），第三个参数是可选的。如果没有传递第三个参数，其默认步长为 1。

GENERATE\_SERIES 函数非常灵活，因此，不必对参数进行硬编码。例如，如果要返回从值 10 开始至 30 结束的 5 行信息，那么递增步长为 5，因此结果集如下所示：

```
ID
---
10
15
20
25
30
```

可以发挥自己的创造性，编写类似这样的查询：

```
select id
  from generate_series(
    (select min(deptno) from emp),
    (select max(deptno) from emp),
    5
  ) x(id)
```

这里应该注意的是，当编写该查询时，并不知道传递给 GENERATE\_SERIES 的实际值，它们是执行主查询时由子查询生成的。

# 高级查找

有一种非常真实的感觉就是：迄今为止，本书一直都在介绍查找。本书中介绍了所有种类的查询，这些查询使用联接和 WHERE 子句，我们还讲解了分组技巧，用于进行查找并返回想要的结果。有些类型的查找操作不同于其他查找操作，它们代表查找的另一种思路。也许需要在显示结果集时，一次只显示一页。问题的前半部分是要确定（查找）要显示的整个记录集，问题另一半是当用户在界面上翻页时，反复查找要显示的下一页记录。分页不是查找问题，但是，依照这种思路，就可能会解决这个问题；本章将介绍这种类型的查找解决方案。

## 11.1 给结果集分页

### 问题

给结果集分页或“滚动显示”整个结果集。例如，先返回表 EMP 中的前 5 行工资，然后是下 5 行，依此类推。目标是允许用户一次能够查看 5 个记录，每次单击“下一页”按钮都会向前滚动 5 个记录。

### 解决方案

在 SQL 中，由于没有“第一个”、“最后一个”及“下一个”的概念，所以必须对要处理的行按某种方式排序，只有如此，才会准确地返回一定范围内记录。

#### DB2、Oracle 和 SQL Server

使用窗口函数 ROW\_NUMBER OVER，可以据此排序，而且能够在 WHERE 子句中指定要返回的记录窗口。例如，要返回 1~5 行：

```
select sal
  from (
select row_number() over (order by sal) as rn,
       sal
  from emp
 ) x
```

```
where rn between 1 and 5
SAL
-----
800
950
1100
1250
1250
```

然后返回 6~10 行:

```
select sal
  from (
select row_number() over (order by sal) as rn,
       sal
  from emp
 ) x
 where rn between 6 and 10
SAL
-----
1300
1500
1600
2450
2850
```

只要修改查询中的 WHERE 子句, 就能够返回任意范围的行。

## MySQL 和 PostgreSQL

由于这两种产品支持 LIMIT 和 OFFSET 子句, 所以滚动结果集相当容易。可使用 LIMIT 指定要返回的行数, 使用 OFFSET 指定要跳过的行数。例如, 按工资的顺序返回前 5 行:

```
select sal
  from emp
 order by sal limit 5 offset 0
SAL
-----
800
950
1100
1250
1250
```

要返回下一组 5 行:

```
select sal
  from emp
 order by sal limit 5 offset 5
SAL
-----
1300
1500
1600
2450
2850
```

LIMIT 和 OFFSET 不仅使 MySQL 和 PostgreSQL 的解决方案容易编写, 而且它们的可读性也很好。



## 讨论

### DB2、Oracle 和 SQL Server

内联视图 X 中的窗口函数 ROW\_NUMBER OVER 给每个工资分配一个唯一的序号（从 1 开始递增）。下面列出了内联视图 X 的结果集：

```
select row_number() over (order by sal) as rn,  
       sal  
from emp
```

RN	SAL
1	800
2	950
3	1100
4	1250
5	1250
6	1300
7	1500
8	1600
9	2450
10	2850
11	2975
12	3000
13	3000
14	5000

把一个序号分配给某个工资之后，只需指定 RN 的范围，就可以选择出要返回的范围。

Oracle 用户还有另外一种方案：可以使用 ROWNUM 代替 ROW NUMBER OVER 生成行的序列号：

```
select sal  
from (  
  select sal, rownum rn  
  from (  
    select sal  
    from emp  
    order by sal  
  )  
  )  
where rn between 6 and 10
```

SAL
1300
1500
1600
2450
2850

如果使用 ROWNUM，那么会要求程序员多编写一层子查询。最里面的子查询是按工资给行排序，它外面的子查询给这些行编序号，最外面的 SELECT 会返回要查找的数据。

### MySQL 和 PostgreSQL

在 SELECT 子句中加入 OFFSET 子句，使滚动结果的操作既直观又容易。指定 OFFSET 为 0 表示从第 1 行开始，OFFSET 为 5 表示从第 6 行开始，OFFSET 为 10 表示从第 11 行

开始。LIMIT 子句限定要返回的行数。把两个子句联合起来使用，就能够很容易地指定从结果集的哪一行开始，要返回多少行。

## 11.2 跳过表中 n 行

### 问题

编写一个查询，从表 EMP 中每隔一行返回一名员工；需要查找第 1 个员工、第 3 个员工，以此类推。例如，从下列结果集中：

```
ENAME
-----
ADAMS
ALLEN
BLAKE
CLARK
FORD
JAMES
JONES
KING
MARTIN
MILLER
SCOTT
SMITH
TURNER
WARD
```

要返回：

```
ENAME
-----
ADAMS
BLAKE
FORD
JONES
MARTIN
SCOTT
TURNER
```

### 解决方案

要跳过结果集中的第 2 行、第 4 行或第 n 行，必须给结果集排序，否则就没有第 1、第 2、下一个或第 4 的概念。

#### DB2、Oracle 和 SQL Server

使用窗口函数 ROW\_NUMBER OVER 给每一行分配一个序号，然后它与求模函数一起使用，就可以跳过不想要的行。在 DB2 和 Oracle 中，求模函数是 MOD。而在 SQL Server 中，使用百分号 (%) 运算符。下面的例子使用 MOD 跳过序号为偶数的行：

```
1 select ename
2   from (
3 select row_number() over (order by ename) rn,
4        ename
5   from emp
6   ) x
```

```
7 where mod(rn,2) = 1
```

## MySQL 和 PostgreSQL

由于它们没有为行分等级或给行编号的内置函数，所以需要使用标量子查询给行分等级（下面的例子是按名字分级的）。然后使用求模操作跳过行：

```
1 select x.ename
2   from (
3   select a.ename,
4          (select count(*)
5           from emp b
6           where b.ename <= a.ename) as rn
7   from emp a
8   ) x
9  where mod(x.rn,2) = 1
```

## 讨论

### DB2、Oracle 和 SQL Server

在内联视图 X 中，调用窗口函数 ROW\_NUMBER OVER，会给每行分配一个等级（不捆绑，姓名重复也不影响。捆绑指对给定的列，值相同的记录等级也相同；不捆绑表示不管记录的值，每行的等级都不同——译者注）。结果集如下所示：

```
select row_number() over (order by ename) rn, ename
from emp
```

```
RN ENAME
-----
1 ADAMS
2 ALLEN
3 BLAKE
4 CLARK
5 FORD
6 JAMES
7 JONES
8 KING
9 MARTIN
10 MILLER
11 SCOTT
12 SMITH
13 TURNER
14 WARD
```

最后，使用求模操作跳过不满足条件的行。

## MySQL 和 PostgreSQL

使用函数给行编号或分等级，之后，可以使用标量子查询，给员工名分等级。内联视图 X 给每个姓名分等级，如下所示：

```
select a.ename,
       (select count(*)
        from emp b
        where b.ename <= a.ename) as rn
from emp a
```

ENAME	RN
-----	-----
ADAMS	1
ALLEN	2
BLAKE	3
CLARK	4
FORD	5
JAMES	6
JONES	7
KING	8
MARTIN	9
MILLER	10
SCOTT	11
SMITH	12
TURNER	13
WARD	14

最后，使用求模函数跳过不满足条件的那些行。

### 11.3 在外联接中用 OR 逻辑

#### 问题

返回部门 10 和 20 中所有员工的姓名和部门信息，并返回部门 30 和 40（但不包含员工信息）的部门信息。开始可能会这么做：

```
select e.ename, d.deptno, d.dname, d.loc
  from dept d, emp e
 where d.deptno = e.deptno
    and (e.deptno = 10 or e.deptno = 20)
 order by 2
```

ENAME	DEPTNO	DNAME	LOC
-----	-----	-----	-----
CLARK	10	ACCOUNTING	NEW YORK
KING	10	ACCOUNTING	NEW YORK
MILLER	10	ACCOUNTING	NEW YORK
SMITH	20	RESEARCH	DALLAS
ADAMS	20	RESEARCH	DALLAS
FORD	20	RESEARCH	DALLAS
SCOTT	20	RESEARCH	DALLAS
JONES	20	RESEARCH	DALLAS

可以看到，因为这个查询中使用的是内部联接，所以结果集不会包含 DEPTNO 30 和 40 的部门信息。

下面的查询中，试图把 EMP 与 DEPT 进行外部联接，但还是不能得到正确结果：

```
select e.ename, d.deptno, d.dname, d.loc
  from dept d left join emp e
    on (d.deptno = e.deptno)
 where e.deptno = 10
    or e.deptno = 20
 order by 2
```

ENAME	DEPTNO	DNAME	LOC
-----	-----	-----	-----
CLARK	10	ACCOUNTING	NEW YORK
KING	10	ACCOUNTING	NEW YORK
MILLER	10	ACCOUNTING	NEW YORK
SMITH	20	RESEARCH	DALLAS
ADAMS	20	RESEARCH	DALLAS
FORD	20	RESEARCH	DALLAS

SCOTT	20	RESEARCH	DALLAS
JONES	20	RESEARCH	DALLAS

最终想要的是下列结果集：

ENAME	DEPTNO	DNAME	LOC
CLARK	10	ACCOUNTING	NEW YORK
KING	10	ACCOUNTING	NEW YORK
MILLER	10	ACCOUNTING	NEW YORK
SMITH	20	RESEARCH	DALLAS
JONES	20	RESEARCH	DALLAS
SCOTT	20	RESEARCH	DALLAS
ADAMS	20	RESEARCH	DALLAS
FORD	20	RESEARCH	DALLAS
	30	SALES	CHICAGO
	40	OPERATIONS	BOSTON

## 解决方案

### DB2、MySQL、PostgreSQL 和 SQL Server

将 OR 条件移到 JOIN 子句中：

```
1 select e.ename, d.deptno, d.dname, d.loc
2   from dept d left join emp e
3     on (d.deptno = e.deptno
4        and (e.deptno=10 or e.deptno=20))
5  order by 2
```

另外，还可以先用 EMP.DEPTNO 进行筛选，然后进行外部联接：

```
1 select e.ename, d.deptno, d.dname, d.loc
2   from dept d
3  left join
4    (select ename, deptno
5     from emp
6     where deptno in ( 10, 20 )
7     ) e on ( e.deptno = d.deptno )
8  order by 2
```

### Oracle

对于 Oracle9i Database 或更高版本，其他产品的两种解决方案都适用；较早的版本中必须使用 CASE 或 DECODE。下面给出了使用 CASE 的解决方案：

```
select e.ename, d.deptno, d.dname, d.loc
  from dept d, emp e
 where d.deptno = e.deptno (+)
    and d.deptno = case when e.deptno(+) = 10 then e.deptno(+)
                       when e.deptno(+) = 20 then e.deptno(+)
                       end
 order by 2
```

下面的解决方案同上，只是使用了 DECODE：

```
select e.ename, d.deptno, d.dname, d.loc
  from dept d, emp e
 where d.deptno = e.deptno (+)
    and d.deptno = decode(e.deptno(+),10,e.deptno(+),
                        20,e.deptno(+))
 order by 2
```

在外部联接列中，如果采用 Oracle 特有的外部联接语法 (+)，而且其中使用 IN 或 OR 谓词，该查询就会出错误，其解决方案是将 IN 或 OR 谓词移到内联视图中：

```
select e.ename, d.deptno, d.dname, d.loc
  from dept d,
       ( select ename, deptno
         from emp
        where deptno in ( 10, 20 )
       ) e
 where d.deptno = e.deptno (+)
 order by 2
```

## 讨论

### DB2、MySQL、PostgreSQL 和 SQL Server

对于这些产品，上面讲述了两两种解决方案。第一种方案是把 OR 条件移到 JOIN 子句中，使它成为联接条件的一部分。这样做之后，就可以筛选出从 EMP 返回的行，而不会丢失 DEPT 中的 DEPTNO 30 和 40。

第二种解决方案是把筛选移到内联视图中。内联视图 E 依据 EMP.DEPTNO 进行筛选，并返回 EMP 中所想要的行。然后，把它们与 DEPT 进行外部联接。在外部联接中，由于 DEPT 是锚表 (anchor table)，因此会返回所有部门，其中也包括 30 和 40。

### Oracle

较老的外部联接语法中似乎有些缺陷，用 CASE 和 DECODE 函数作为应对措施。本解决方案使用了内联视图 E，它先找到表 EMP 中感兴趣的行，然后与 DEPT 进行外部联接。

## 11.4 确定哪些行是彼此互换的

### 问题

一个表中包含两次考试的结果，要确定哪些分数对是互换的。下面列出了视图 V 的结果集：

```
select *
  from V
TEST1      TEST2
-----
20          20
50          25
20          20
60          30
70          90
80          130
90          70
100         50
110         55
120         60
130         80
140         70
```

检验这些结果，会发现 TEST1 70 和 TEST2 90 的考试分数是互换的（存在 TEST1 90 和 TEST2 70 的行），同样，TEST1 80 和 TEST2 130 的分数与 TEST1 130 和 TEST2 80 也是互换的，另外，TEST1 20 和 TEST2 20 的分数与 TEST1 20 和 TEST2 20 是互换的。要求每组互换行仅找出一个互换对，其结果集应如：

TEST1	TEST2
20	20
70	90
80	130

而不是：

TEST1	TEST2
20	20
20	20
70	90
80	130
90	70
130	80

## 解决方案

使用自联接，识别 TEST1 等于 TEST2（反之亦然）的行：

```
select distinct v1.*
  from V v1, V v2
 where v1.test1 = v2.test2
    and v1.test2 = v2.test1
    and v1.test1 <= v1.test2
```

## 讨论

自联接的结果是笛卡儿积，可以把每个 TEST1 分数与所有 TEST2 分数相比较，反之亦然。下面的查询将返回互换数：

```
select v1.*
  from V v1, V v2
 where v1.test1 = v2.test2
    and v1.test2 = v2.test1
```

TEST1	TEST2
20	20
20	20
20	20
20	20
20	20
90	70
130	80
70	90
80	130

使用 DISTINCT，可以确保从最终结果集中删除重复行。WHERE 子句的最后一个筛选（and V1.TEST1 <= V1.TEST2）确保只返回一对互换数（在此，TEST1 是较小值或相等值）。

## 11.5 选择前 n 个记录

### 问题

以某种排序方式，限定结果集只包含一定数目的记录。例如，返回最高 5 档工资的员工姓名和工资。

### 解决方案

这种解决方案的关键是两个步骤：首先按预定方式给行排序，然后限定结果集，只包含感兴趣的行。

#### DB2、Oracle 和 SQL Server

这个问题的解决方案取决于窗口函数的使用。使用什么窗口函数取决于如何处理捆绑。下面的解决方案使用了 DENSE\_RANK，这样，在总计中，工资的每次捆绑都只计算一次：

```
1 select ename,sal
2   from (
3 select ename, sal,
4        dense_rank() over (order by sal desc) dr
5   from emp
6  ) x
7  where dr <= 5
```

返回的总行数可能会超过 5，但只有 5 个不同的工资。如果希望返回 5 行而不管捆绑的话，可使用 ROW\_NUMBER OVER（因为这个函数不允许捆绑）。

#### MySQL 和 PostgreSQL

使用标量子查询，为每个工资创建一个等级。然后利用等级限制子查询的结果：

```
1 select ename,sal
2   from (
3 select (select count(distinct b.sal)
4        from emp b
5        where a.sal <= b.sal) as rnk,
6        a.sal,
7        a.ename
8   from emp a
9  )
10  where rnk <= 5
```

### 讨论

#### DB2、Oracle 和 SQL Server

内联视图 X 中的窗口函数 DENSE\_RANK OVER 完成所有这些操作。下面的例子显示了使用该函数之后返回的整个表：

```
select ename, sal,
       dense_rank() over (order by sal desc) dr
  from emp
```



ENAME	SAL	DR
KING	5000	1
SCOTT	3000	2
FORD	3000	2
JONES	2975	3
BLAKE	2850	4
CLARK	2450	5
ALLEN	1600	6
TURNER	1500	7
MILLER	1300	8
WARD	1250	9
MARTIN	1250	9
ADAMS	1100	10
JAMES	950	11
SMITH	800	12

现在需要做的是返回 DR 小于等于 5 的所有行。

## MySQL 和 PostgreSQL

内联视图 X 中的标量子查询用于给工资分等级，如下所示：

```
select (select count(distinct b.sal)
        from emp b
        where a.sal <= b.sal) as rnk,
       a.sal,
       a.ename
from emp a
```

RNK	SAL	ENAME
1	5000	KING
2	3000	SCOTT
2	3000	FORD
3	2975	JONES
4	2850	BLAKE
5	2450	CLARK
6	1600	ALLEN
7	1500	TURNER
8	1300	MILLER
9	1250	WARD
9	1250	MARTIN
10	1100	ADAMS
11	950	JAMES
12	800	SMITH

最后一步，只返回 RNK 小于等于 5 的行。

## 11.6 找到包含最大值和最小值的记录

### 问题

查找表中的“两极”值。例如，找出表 EMP 中具有最高工资和最低工资的员工。

### 解决方案

DB2、Oracle 和 SQL Server

使用窗口函数 MIN OVER 和 MAX OVER，在 TMP 表中分别找到最低工资和最高工资：

```

1 select ename
2   from (
3 select ename, sal,
4        min(sal)over() min_sal,
5        max(sal)over() max_sal
6   from emp
7        ) x
8  where sal in (min_sal,max_sal)

```

## MySQL 和 PostgreSQL

编写两个查询，分别用于返回 SAL 的 MIN 和 MAX 值：

```

1 select ename
2   from emp
3  where sal in ( (select min(sal) from emp),
4                (select max(sal) from emp) )

```

## 讨论

### DB2、Oracle 和 SQL Server

使用窗口函数 MIN OVER 和 MAX OVER 可以每行访问最低工资和最高工资。内联视图 X 的结果集如下所示：

```

select ename, sal,
       min(sal)over() min_sal,
       max(sal)over() max_sal
from emp

```

ENAME	SAL	MIN_SAL	MAX_SAL
SMITH	800	800	5000
ALLEN	1600	800	5000
WARD	1250	800	5000
JONES	2975	800	5000
MARTIN	1250	800	5000
BLAKE	2850	800	5000
CLARK	2450	800	5000
SCOTT	3000	800	5000
KING	5000	800	5000
TURNER	1500	800	5000
ADAMS	1100	800	5000
JAMES	950	800	5000
FORD	3000	800	5000
MILLER	1300	800	5000

得到这个结果集之后，接下来要返回 SAL 等于 MIN\_SAL 或 MAX\_SAL 的行。

## MySQL 和 PostgreSQL

该解决方案在一个 IN 列表中使用两个子查询，分别用于从 EMP 中找到最低工资和最高工资。由外层查询返回工资与其中一个子查询返回的值相匹配的行。

## 11.7 存取“未来”行

### 问题

找到满足这样条件的员工：即他的收入比紧随其后聘用的员工要少。基于下列结果集：

ENAME	SAL	HIREDATE
SMITH	800	17-DEC-80
ALLEN	1600	20-FEB-81
WARD	1250	22-FEB-81
JONES	2975	02-APR-81
BLAKE	2850	01-MAY-81
CLARK	2450	09-JUN-81
TURNER	1500	08-SEP-81
MARTIN	1250	28-SEP-81
KING	5000	17-NOV-81
JAMES	950	03-DEC-81
FORD	3000	03-DEC-81
MILLER	1300	23-JAN-82
SCOTT	3000	09-DEC-82
ADAMS	1100	12-JAN-83

SMITH、WARD、MARTIN、JAMES 和 MILLER 的收入都比紧随其后聘用的员工要低，因此他们就是希望使用查询找到的员工。

## 解决方案

首先定义“未来”的意思。必须给结果集排序，才能够定义一个值在另一个值“后面”。

### DB2、MySQL、PostgreSQL 和 SQL Server

使用子查询，确定每个员工的下列信息：

- 在他之后聘用的第一个工资高于他的员工的聘用日期
- 紧随其后聘用的下一个人的日期

当两个日期相匹配时，就得到了要查找的内容：

```
1 select ename, sal, hiredate
2   from (
3 select a.ename, a.sal, a.hiredate,
4        (select min(hiredate) from emp b
5         where b.hiredate > a.hiredate
6         and b.sal > a.sal ) as next_sal_grtr,
7        (select min(hiredate) from emp b
8         where b.hiredate > a.hiredate) as next_hire
9   from emp a
10  ) x
11  where next_sal_grtr = next_hire
```

### Oracle

可以使用 LEAD OVER 窗口函数，访问下一个聘用员工的工资，接下来检查工资是否高于他就相当容易了：

```
1 select ename, sal, hiredate
2   from (
3 select ename, sal, hiredate,
4        lead(sal)over(order by hiredate) next_sal
5   from emp
6  )
7  where sal < next_sal
```

## 讨论

### DB2、MySQL、PostgreSQL 和 SQL Server

标量子查询会为每个员工返回紧随其后聘用员工的 HIREDATE, 以及在其后聘用且比当前员工收入高的第一个员工的 HIREDATE。下面列出了未处理的原始数据:

```
select a.ename, a.sal, a.hiredate,
       (select min(hiredate) from emp b
        where b.hiredate > a.hiredate
          and b.sal > a.sal ) as next_sal_grtr,
       (select min(hiredate) from emp b
        where b.hiredate > a.hiredate) as next_hire
from emp a
```

ENAME	SAL	HIREDATE	NEXT_SAL_GRTR	NEXT_HIRE
SMITH	800	17-DEC-80	20-FEB-81	20-FEB-81
ALLEN	1600	20-FEB-81	02-APR-81	22-FEB-81
WARD	1250	22-FEB-81	02-APR-81	02-APR-81
JONES	2975	02-APR-81	17-NOV-81	01-MAY-81
MARTIN	1250	28-SEP-81	17-NOV-81	17-NOV-81
BLAKE	2850	01-MAY-81	17-NOV-81	09-JUN-81
CLARK	2450	09-JUN-81	17-NOV-81	08-SEP-81
SCOTT	3000	09-DEC-82		12-JAN-83
KING	5000	17-NOV-81		03-DEC-81
TURNER	1500	08-SEP-81	17-NOV-81	28-SEP-81
ADAMS	1100	12-JAN-83		
JAMES	950	03-DEC-81	23-JAN-82	23-JAN-82
FORD	3000	03-DEC-81		23-JAN-82
MILLER	1300	23-JAN-82	09-DEC-82	09-DEC-82

后来聘用的员工可能紧随当前员工之后, 也可能不是。那么, 下一步 (最后一步), 只返回 NEXT\_SAL\_GRTR (收入比当前员工高的最早的 HIREDATE) 等于 NEXT\_HIRE (与当前员工相邻的下一个员工的 HIREDATE) 的行。

## Oracle

解决这类问题, 窗口函数 LEAD OVER 是很理想的。它提供的查询比其他产品的解决方案具有更好的可读性, 而且 LEAD OVER 也使解决方案更为灵活, 这是由于可以给 LEAD OVER 传递一个参数, 用于确定应该向前查看多少行 (默认为 1)。如果正在排序的列存在重复, 那么, 能够向前跳过多行就很重要了。

从下面的例子可以看出, 使用 LEAD OVER 查看 “下一个” 聘用员工的工资多么容易:

```
select ename, sal, hiredate,
       lead(sal)over(order by hiredate) next_sal
from emp
```

ENAME	SAL	HIREDATE	NEXT_SAL
SMITH	800	17-DEC-80	1600
ALLEN	1600	20-FEB-81	1250
WARD	1250	22-FEB-81	2975
JONES	2975	02-APR-81	2850
BLAKE	2850	01-MAY-81	2450
CLARK	2450	09-JUN-81	1500
TURNER	1500	08-SEP-81	1250

MARTIN	1250	28-SEP-81	5000
KING	5000	17-NOV-81	950
JAMES	950	03-DEC-81	3000
FORD	3000	03-DEC-81	1300
MILLER	1300	23-JAN-82	3000
SCOTT	3000	09-DEC-82	1100
ADAMS	1100	12-JAN-83	

最后一步，返回 SAL 小于 NEXT\_SAL 的行。由于 LEAD OVER 默认范围是 1 行，如果表 EMP 中的记录存在重复，特别是同一天聘用了多个员工，那么他们的 SAL 都会拿来比较。这可能是我们想要的结果，也可能不是。如果目标是把每个员工的 SAL 与下一个员工的 SAL 进行比较，而把同一天聘用的其他员工排除在外，那么可使用下列解决方案：

```
select ename, sal, hiredate
  from (
select ename, sal, hiredate,
       lead(sal,cnt-rn+1)over(order by hiredate) next_sal
  from (
select ename,sal,hiredate,
       count(*)over(partition by hiredate) cnt,
       row_number()over(partition by hiredate order by empno) rn
  from emp
  )
  )
 where sal < next_sal
```

然后，找到当前行与它应该比较的行之间的距离。例如，如果存在 5 个重复，那么第一行就需要跳过 5 行，才能到达正确的 LEAD OVER 行。对于 HIREDATE 与其他人重复的每个员工，CNT 值表示总共有多少个重复，RN 值表示该员工在与他 HIREDATE 相同的员工组中的序号，该序号是按 HIREDATE 分组的，因此只有 HIREDATE 重复的员工，其值才会大于 1。序号是按 EMPNO 排定的（只是随意设置）。现在已经知道了总的重复个数，而且每个重复都有一个序号，那么与下一个 HIREDATE 的距离就是重复总数减去当前等级再加 1 (CNT-RN+1)。

## 参阅

对于使用 LEAD OVER 处理重复的其他例子（有关上述技巧的更多讨论），请参阅第 8 章中的 8.7 节及第 10 章中 10.2 节。

## 11.8 轮换行值

### 问题

返回每个员工的姓名和工资以及低于自己的最高工资和高于自己的最低工资。如果没有更高或更低的工资，则可能要求结果环绕（即第一个 SAL 显示最后一个 SAL，反之亦然）。应返回下列结果集：

ENAME	SAL	FORWARD	REWIND
SMITH	800	950	5000
JAMES	950	1100	800
ADAMS	1100	1250	950

WARD	1250	1250	1100
MARTIN	1250	1300	1250
MILLER	1300	1500	1250
TURNER	1500	1600	1300
ALLEN	1600	2450	1500
CLARK	2450	2850	1600
BLAKE	2850	2975	2450
JONES	2975	3000	2850
SCOTT	3000	3000	2975
FORD	3000	5000	3000
KING	5000	800	3000

## 解决方案

对于 Oracle 用户，窗口函数 LEAD OVER 和 LAG OVER 使这个问题相当容易解决，而且其查询具有很高的可读性。对于其他 RDBMS，可以使用标量子查询，但捆绑会带来问题，正是鉴于这种原因，在不支持窗口函数 RDBMS 中，对于这个问题只能提供近似的解决方案。

### DB2、SQL Server、MySQL 和 PostgreSQL

使用标量子查询，找到相对于每个工资的下一档工资和上一档工资：

```

1  select e.ename, e.sal,
2         coalesce(
3             (select min(sal) from emp d where d.sal > e.sal),
4             (select min(sal) from emp)
5         ) as forward,
6         coalesce(
7             (select max(sal) from emp d where d.sal < e.sal),
8             (select max(sal) from emp)
9         ) as rewind
10 from emp e
11 order by 2
```

### Oracle

使用窗口函数 LAG OVER 和 LEAD OVER，访问相对于当前行的下一行和上一行：

```

1 select ename, sal,
2        nvl(lead(sal)over(order by sal),min(sal)over()) forward,
3        nvl(lag(sal)over(order by sal),max(sal)over()) rewind
4 from emp
```

## 讨论

### DB2、SQL Server、MySQL 和 PostgreSQL

标量子查询解决方案并不是这个问题的真正解决方案。这只是个近似方案，一旦两个记录包含同样的 SAL 值就会失败。如果不能使用窗口函数，这就是最好的解决方案了。

### Oracle

窗口函数 LAG OVER 和 LEAD OVER 分别返回当前行的前一行及下一行（默认如此，除非另行指定）的值。OVER 子句的 ORDER BY 中定义了“前一个”和“下一个”的含义。如果检验该解决方案，第一步会按 SAL 的顺序返回相对于当前行的下一行和上一行：

```
select ename,sal,
       lead(sal)over(order by sal) forward,
       lag(sal)over(order by sal) rewind
from emp
```

ENAME	SAL	FORWARD	REWIND
SMITH	800	950	
JAMES	950	1100	800
ADAMS	1100	1250	950
WARD	1250	1250	1100
MARTIN	1250	1300	1250
MILLER	1300	1500	1250
TURNER	1500	1600	1300
ALLEN	1600	2450	1500
CLARK	2450	2850	1600
BLAKE	2850	2975	2450
JONES	2975	3000	2850
SCOTT	3000	3000	2975
FORD	3000	5000	3000
KING	5000		3000

注意，员工 SMITH 的 REWIND 为 NULL 值，而员工 KING 的 FORWARD 为 NULL 值，其原因是这两个员工分别为最高工资和最低工资。针对“问题”一节中提出的需求，当 FORWARD 或 REWIND 中存在 NULL 值时，应该“环绕”结果，也就是说，对于最高的 SAL，其 FORWARD 应该是表中最底的 SAL 值；而对于最低的 SAL，其 REWIND 应该是表中最高的 SAL 值。对于窗口函数 MIN OVER 和 MAX OVER，如果未指定分区或窗口（即 OVER 子句后面为空值），那么它们将分别返回表中的最低工资和最高工资。结果集如下所示：

```
select ename,sal,
       nvl(lead(sal)over(order by sal),min(sal)over()) forward,
       nvl(lag(sal)over(order by sal),max(sal)over()) rewind
from emp
```

ENAME	SAL	FORWARD	REWIND
SMITH	800	950	5000
JAMES	950	1100	800
ADAMS	1100	1250	950
WARD	1250	1250	1100
MARTIN	1250	1300	1250
MILLER	1300	1500	1250
TURNER	1500	1600	1300
ALLEN	1600	2450	1500
CLARK	2450	2850	1600
BLAKE	2850	2975	2450
JONES	2975	3000	2850
SCOTT	3000	3000	2975
FORD	3000	5000	3000
KING	5000	800	3000

LAG OVER 和 LEAD OVER 的另一个有用特征是：能够定义向前或向后移动多远。在本节的例子中，只向前或向后移动 1 行。如果想向前移动 3 行、向后移动 5 行，也是非常容易实现的。只需用 3 和 5 作为参数分别传递给 LEAD 和 LAG 即可，如下所示：

```
select ename,sal,
       lead(sal,3)over(order by sal) forward,
       lag(sal,5)over(order by sal) rewind
from emp
```

ENAME	SAL	FORWARD	REWIND
SMITH	800	1250	
JAMES	950	1250	
ADAMS	1100	1300	
WARD	1250	1500	
MARTIN	1250	1600	
MILLER	1300	2450	800
TURNER	1500	2850	950
ALLEN	1600	2975	1100
CLARK	2450	3000	1250
BLAKE	2850	3000	1250
JONES	2975	5000	1300
SCOTT	3000		1500
FORD	3000		1600
KING	5000		2450

## 11.9 给结果分等级

### 问题

给表 EMP 中的工资分等级，并允许捆绑，返回下列结果集：

RNK	SAL
1	800
2	950
3	1100
4	1250
4	1250
5	1300
6	1500
7	1600
8	2450
9	2850
10	2975
11	3000
11	3000
12	5000

### 解决方案

窗口函数会使等级查询相当简单。要进行分等级，下面三个窗口函数特别有用：DENSE\_RANK OVER, ROW\_NUMBER OVER 和 RANK OVER.

#### DB2、Oracle 和 SQL Server

因为允许捆绑，所以可使用窗口函数 DENSE\_RANK OVER:

```
1 select dense_rank() over(order by sal) rnk, sal
2   from emp
```

#### MySQL and PostgreSQL

在提供窗口函数之前，可以使用标量子查询给工资分等级：

```
1 select (select count(distinct b.sal)
2         from emp b
3         where b.sal <= a.sal) as rnk,
4        a.sal
5   from emp a
```



## 讨论

### DB2、Oracle 和 SQL Server

这里，窗口函数 DENSE\_RANK OVER 完成了所有收集信息的工作。在 OVER 关键字后面的圆括号中，加了一个 ORDER BY 子句，用于指定顺序，行会按这种顺序分等级。该解决方案使用了 ORDER BY SAL，所以 EMP 表中的行是按工资的升序分等级的。

### MySQL 和 PostgreSQL

标量子查询解决方案的输出与 DENSE\_RANK 的输出相似，这是由于标量子查询中的驱动谓词是以 SAL 为依据的。

## 11.10 抑制重复

### 问题

在表 EMP 中查找不同的职位，但不想看到有重复。其结果集应该是：

```
JOB
-----
ANALYST
CLERK
MANAGER
PRESIDENT
SALESMAN
```

### 解决方案

所有的 RDBMS 都支持 DISTINCT 关键字，而且在抑制结果集出现重复方面，它无疑是最简单的一种机制。然而，本节也会介绍另外两种抑制重复的方法。

### DB2、Oracle 和 SQL Server

对于这些 RDBMS，无疑可以使用传统的 DISTINCT 方法，有时也可以用 GROUP BY (跟 MySQL/PostgreSQL 解决方案中一样)。以下解决方案用了另一种方法，使用窗口函数 ROW\_NUMBER OVER：

```
1 select job
2   from (
3 select job,
4        row_number()over(partition by job order by job) rn
5   from emp
6   ) x
7  where rn = 1
```

### MySQL 和 PostgreSQL

使用 DISTINCT 关键字可以抑制结果集中的重复：

```
select distinct job
  from emp
```

另外，也可以使用 GROUP BY 抑制重复

```
select job
  from emp
 group by job
```

## 讨论

### DB2、Oracle 和 SQL Server

这个解决方案依赖一些有关分区窗口函数的非传统思维。在 ROW\_NUMBER 的 OVER 子句中使用 PARTITION BY，每当遇到新职位时就把 ROW\_NUMBER 的返回值重置为 1。下面给出了内联视图 X 的结果集：

```
select job,
       row_number()over(partition by job order by job) rn
  from emp
```

JOB	RN
ANALYST	1
ANALYST	2
CLERK	1
CLERK	2
CLERK	3
CLERK	4
MANAGER	1
MANAGER	2
MANAGER	3
PRESIDENT	1
SALESMAN	1
SALESMAN	2
SALESMAN	3
SALESMAN	4

每行都分配到一个递增的序号，每当职位改变时，这个号码就会重置为 1。要筛选掉重复项，只需要保留 RN 为 1 的行。

在使用 ROW\_NUMBER OVER 时，必须使用 ORDER BY 子句（除 DB2 之外），但它不会影响结果。只要对每个职位都返回一行，具体返回哪一行无关紧要。

### MySQL 和 PostgreSQL

最第一种解决方案中，展示了如何使用关键字 DISTINCT 抑制结果集中的重复行。记住 DISTINCT 会应用于整个 SELECT 列表；其他列可能会更改结果集。想一想下面两个查询之间的差别：

```
select distinct job
  from emp
```

JOB
ANALYST
CLERK
MANAGER
PRESIDENT
SALESMAN

```
select distinct job, deptno
  from emp
```

JOB	DEPTNO
ANALYST	20
CLERK	10
CLERK	20
CLERK	30
MANAGER	10
MANAGER	20
MANAGER	30
PRESIDENT	10

SALESMAN

30

在 SELECT 列表中加入 DEPTNO 后, 返回的内容是表 EMP 中各不相同的 JOB/DEPTNO 值对。

第二种解决方案使用 GROUP BY 抑制重复。按这种方式使用 GROUP BY 并不常见, 记住, GROUP BY 和 DISTINCT 是两种不同的子句, 两者不能互换。这里介绍 GROUP BY 方案是出于内容完整的考虑, 无疑会在有些情况下碰到这种用法。

## 11.11 找到骑士值

### 问题

返回一个结果集, 它包含每个部门中所有员工的姓名、所在部门、工资、聘用日期以及部门中最新聘用员工的工资。应返回下列结果集:

DEPTNO	ENAME	SAL	HIREDATE	LATEST_SAL
10	MILLER	1300	23-JAN-1982	1300
10	KING	5000	17-NOV-1981	1300
10	CLARK	2450	09-JUN-1981	1300
20	ADAMS	1100	12-JAN-1983	1100
20	SCOTT	3000	09-DEC-1982	1100
20	FORD	3000	03-DEC-1981	1100
20	JONES	2975	02-APR-1981	1100
20	SMITH	800	17-DEC-1980	1100
30	JAMES	950	03-DEC-1981	950
30	MARTIN	1250	28-SEP-1981	950
30	TURNER	1500	08-SEP-1981	950
30	BLAKE	2850	01-MAY-1981	950
30	WARD	1250	22-FEB-1981	950
30	ALLEN	1600	20-FEB-1981	950

LATEST\_SAL 中的值是“骑士值”, 这是由于查找它们的路径与国际象棋中骑士(马)的路径相似, 求结果的方法跟骑士走到新位置的方式一样: 跳到一行, 然后转向跳到另一列(请参阅图 11-1)。要找到 LATEST\_SAL 的正确值, 必须先定位(跳到)每个 DEPTNO 中最新 HIREDATE 所在的行, 然后, 选择(跳到)该行的 SAL 列。

**注意:** 术语“骑士值”是由我的一位聪明合作者 Kay Young 发明的。让他审查本节正确性之后, 我告诉他遇到难题了, 不能起一个好名称。由于是必须先计算一行, 然后“跳转”, 取另一行的值, 因此他提出了术语“骑士值”。

### 解决方案

#### DB2 和 SQL Server

在子查询中使用一个 CASE 表达式, 返回每个 DEPTNO 中最新聘用的员工的 SAL, 而对于其他工资, 都返回 0。在外层查询中使用窗口函数 MAX OVER, 返回每个员工所在部门的非 0 SAL:

DEPTNO	ENAME	SAL	HIREDATE	LATEST_SAL
30	JAMES	950	03-DEC-1981	950
30	MARTIN	1250	28-SEP-1981	950
30	TURNER	1500	08-SEP-1981	950
30	BLAKE	2850	01-MAY-1981	950
30	WARD	1250	22-FEB-1981	950
30	ALLEN	1600	20-FEB-1981	950

图 11-1：用“向上再拐弯”的方法求得的骑士值

```

1 select deptno,
2        ename,
3        sal,
4        hiredate,
5        max(latest_sal)over(partition by deptno) latest_sal
6   from (
7 select deptno,
8        ename,
9        sal,
10       hiredate,
11       case
12         when hiredate = max(hiredate)over(partition by deptno)
13         then sal else 0
14       end latest_sal
15   from emp
16   ) x
17  order by 1, 4 desc

```

## MySQL 和 PostgreSQL

使用一个嵌套两层的标量子查询。首先，找到每个 DEPTNO 中最后一个员工的 HIREDATE。然后，使用聚合函数 MAX（如果存在重复的话），查找每个 DEPTNO 中新聘用的员工的 SAL：

```

1 select e.deptno,
2        e.ename,
3        e.sal,
4        e.hiredate,
5        (select max(d.sal)
6         from emp d
7         where d.deptno = e.deptno
8         and d.hiredate =
9              (select max(f.hiredate)
10               from emp f
11               where f.deptno = e.deptno)) as latest_sal
12   from emp e
13  order by 1, 4 desc

```

## Oracle

使用窗口函数 MAX OVER，返回每个 DEPTNO 中最高的 SAL。在 KEEP 子句中，使用

函数 DENSE\_RANK 和 LAST，并按 HIREDATE 排序，以返回给定 DEPTNO 中最新 HIREDATE 的最高 SAL:

```

1 select deptno,
2        ename,
3        sal,
4        hiredate,
5        max(sal)
6        keep(dense_rank last order by hiredate)
7        over(partition by deptno) latest_sal
8   from emp
9  order by 1, 4 desc

```

## 讨论

### DB2 和 SQL Server

首先，在 CASE 表达式中使用窗口函数 MAX OVER，找到每个 DEPTNO 中最新聘用的员工。如果某个员工的 HIREDATE 与 MAX OVER 返回的值相匹配，则使用 CASE 表达式返回该员工的 SAL，否则返回 0。其结果集如下所示：

```

select deptno,
       ename,
       sal,
       hiredate,
       case
         when hiredate = max(hiredate)over(partition by deptno)
         then sal else 0
       end latest_sal
  from emp

```

DEPTNO	ENAME	SAL	HIREDATE	LATEST_SAL
10	CLARK	2450	09-JUN-1981	0
10	KING	5000	17-NOV-1981	0
10	MILLER	1300	23-JAN-1982	1300
20	SMITH	800	17-DEC-1980	0
20	ADAMS	1100	12-JAN-1983	1100
20	FORD	3000	03-DEC-1981	0
20	SCOTT	3000	09-DEC-1982	0
20	JONES	2975	02-APR-1981	0
30	ALLEN	1600	20-FEB-1981	0
30	BLAKE	2850	01-MAY-1981	0
30	MARTIN	1250	28-SEP-1981	0
30	JAMES	950	03-DEC-1981	950
30	TURNER	1500	08-SEP-1981	0
30	WARD	1250	22-FEB-1981	0

由于 LATEST\_SAL 值可能是 0，也可能是最新聘用的员工的 SAL，因此可以把上面的查询包入内联视图中，然后再次使用 MAX OVER，但这次是为了返回每个 DEPTNO 的最大非零 LATEST\_SAL:

```

select deptno,
       ename,
       sal,
       hiredate,
       max(latest_sal)over(partition by deptno) latest_sal
  from (
select deptno,
       ename,

```

```

    sal,
    hiredate,
    case
        when hiredate = max(hiredate)over(partition by deptno)
        then sal else 0
    end latest_sal
from emp
) x
order by 1, 4 desc

```

DEPTNO	ENAME	SAL	HIREDATE	LATEST_SAL
10	MILLER	1300	23-JAN-1982	1300
10	KING	5000	17-NOV-1981	1300
10	CLARK	2450	09-JUN-1981	1300
20	ADAMS	1100	12-JAN-1983	1100
20	SCOTT	3000	09-DEC-1982	1100
20	FORD	3000	03-DEC-1981	1100
20	JONES	2975	02-APR-1981	1100
20	SMITH	800	17-DEC-1980	1100
30	JAMES	950	03-DEC-1981	950
30	MARTIN	1250	28-SEP-1981	950
30	TURNER	1500	08-SEP-1981	950
30	BLAKE	2850	01-MAY-1981	950
30	WARD	1250	22-FEB-1981	950
30	ALLEN	1600	20-FEB-1981	950

## MySQL 和 PostgreSQL

第一步，使用标量子查询，找到每个 DEPTNO 中最新聘用的员工的 HIREDATE：

```

select e.deptno,
       e.ename,
       e.sal,
       e.hiredate,
       (select max(f.hiredate)
        from emp f
        where f.deptno = e.deptno) as last_hire
from emp e
order by 1, 4 desc

```

DEPTNO	ENAME	SAL	HIREDATE	LAST_HIRE
10	MILLER	1300	23-JAN-1982	23-JAN-1982
10	KING	5000	17-NOV-1981	23-JAN-1982
10	CLARK	2450	09-JUN-1981	23-JAN-1982
20	ADAMS	1100	12-JAN-1983	12-JAN-1983
20	SCOTT	3000	09-DEC-1982	12-JAN-1983
20	FORD	3000	03-DEC-1981	12-JAN-1983
20	JONES	2975	02-APR-1981	12-JAN-1983
20	SMITH	800	17-DEC-1980	12-JAN-1983
30	JAMES	950	03-DEC-1981	03-DEC-1981
30	MARTIN	1250	28-SEP-1981	03-DEC-1981
30	TURNER	1500	08-SEP-1981	03-DEC-1981
30	BLAKE	2850	01-MAY-1981	03-DEC-1981
30	WARD	1250	22-FEB-1981	03-DEC-1981
30	ALLEN	1600	20-FEB-1981	03-DEC-1981

然后，查找每个 DEPTNO 中于 LAST\_HIRE 日期聘用的员工的 SAL。使用聚合函数 MAX 保留最高值（如果同一天聘用了多个员工的话）：

```

select e.deptno,
       e.ename,
       e.sal,
       e.hiredate,
       (select max(d.sal)

```

```

        from emp d
        where d.deptno = e.deptno
        and d.hiredate =
            (select max(f.hiredate)
             from emp f
             where f.deptno = e.deptno)) as latest_sal
    from emp e
    order by 1, 4 desc

```

DEPTNO	ENAME	SAL	HIREDATE	LATEST_SAL
10	MILLER	1300	23-JAN-1982	1300
10	KING	5000	17-NOV-1981	1300
10	CLARK	2450	09-JUN-1981	1300
20	ADAMS	1100	12-JAN-1983	1100
20	SCOTT	3000	09-DEC-1982	1100
20	FORD	3000	03-DEC-1981	1100
20	JONES	2975	02-APR-1981	1100
20	SMITH	800	17-DEC-1980	1100
30	JAMES	950	03-DEC-1981	950
30	MARTIN	1250	28-SEP-1981	950
30	TURNER	1500	08-SEP-1981	950
30	BLAKE	2850	01-MAY-1981	950
30	WARD	1250	22-FEB-1981	950
30	ALLEN	1600	20-FEB-1981	950

## Oracle

Oracle8i Database 的用户可以采用 DB2 解决方案。而对于 Oracle9i Database 及更高版本的用户，可以使用下面给出的解决方案。Oracle 解决方案的关键是使用了 KEEP 子句。利用 KEEP 子句可以给组/分区的行分等级，也可以处理组中的第一行和最后一行。不使用 KEEP 的解决方案如下所示：

```

select deptno,
       ename,
       sal,
       hiredate,
       max(sal) over(partition by deptno) latest_sal
  from emp
  order by 1, 4 desc

```

DEPTNO	ENAME	SAL	HIREDATE	LATEST_SAL
10	MILLER	1300	23-JAN-1982	5000
10	KING	5000	17-NOV-1981	5000
10	CLARK	2450	09-JUN-1981	5000
20	ADAMS	1100	12-JAN-1983	3000
20	SCOTT	3000	09-DEC-1982	3000
20	FORD	3000	03-DEC-1981	3000
20	JONES	2975	02-APR-1981	3000
20	SMITH	800	17-DEC-1980	3000
30	JAMES	950	03-DEC-1981	2850
30	MARTIN	1250	28-SEP-1981	2850
30	TURNER	1500	08-SEP-1981	2850
30	BLAKE	2850	01-MAY-1981	2850
30	WARD	1250	22-FEB-1981	2850
30	ALLEN	1600	20-FEB-1981	2850

当 MAX OVER 不使用 KEEP 时，就不会返回最新聘用员工的 SAL，而只是返回每个 DEPTNO 中的最高工资。在本节中，通过指定 ORDER BY HIREDATE，可以用 KEEP 子句按 HIREDATE 给每个 DEPTNO 中的工资排序，然后，函数 DENSE\_RANK 按升序给每个 HIREDATE 分配一个等级。最后，函数 LAST 确定要使用聚集函数的行：即基于

DENSE\_RANK 等级的“末尾”一行。在这个例子中，聚集函数 MAX 作用于最后一个 HIREDATE 所在行中的 SAL 列。其实就是保留了每个部门中 HIREDATE 最后一个等级的 SAL。

这里的做法是基于某一列 (HIREDATE) 给每个 DEPTNO 的行分等级，然后对另外一列 (SAL) 使用聚集函数 (MAX)。这种做法非常有用，避免其他解决方案中所额外采用的联接和内联视图。最后，在 KEEP 子句后面加上 OVER 子句，就能够返回 KEEP 为每一行“保留”的 SAL。

此外，也可以按降序的 HIREDATE 进行排序，从而“保留”第一个 SAL。比较下面这两种查询，它们会返回相同的结果集：

```
select deptno,
       ename,
       sal,
       hiredate,
       max(sal)
         keep(dense_rank last order by hiredate)
         over(partition by deptno) latest_sal
  from emp
 order by 1, 4 desc
```

DEPTNO	ENAME	SAL	HIREDATE	LATEST_SAL
10	MILLER	1300	23-JAN-1982	1300
10	KING	5000	17-NOV-1981	1300
10	CLARK	2450	09-JUN-1981	1300
20	ADAMS	1100	12-JAN-1983	1100
20	SCOTT	3000	09-DEC-1982	1100
20	FORD	3000	03-DEC-1981	1100
20	JONES	2975	02-APR-1981	1100
20	SMITH	800	17-DEC-1980	1100
30	JAMES	950	03-DEC-1981	950
30	MARTIN	1250	28-SEP-1981	950
30	TURNER	1500	08-SEP-1981	950
30	BLAKE	2850	01-MAY-1981	950
30	WARD	1250	22-FEB-1981	950
30	ALLEN	1600	20-FEB-1981	950

```
select deptno,
       ename,
       sal,
       hiredate,
       max(sal)
         keep(dense_rank first order by hiredate desc)
         over(partition by deptno) latest_sal
  from emp
 order by 1, 4 desc
```

DEPTNO	ENAME	SAL	HIREDATE	LATEST_SAL
10	MILLER	1300	23-JAN-1982	1300
10	KING	5000	17-NOV-1981	1300
10	CLARK	2450	09-JUN-1981	1300
20	ADAMS	1100	12-JAN-1983	1100
20	SCOTT	3000	09-DEC-1982	1100
20	FORD	3000	03-DEC-1981	1100
20	JONES	2975	02-APR-1981	1100
20	SMITH	800	17-DEC-1980	1100



30 JAMES	950 03-DEC-1981	950
30 MARTIN	1250 28-SEP-1981	950
30 TURNER	1500 08-SEP-1981	950
30 BLAKE	2850 01-MAY-1981	950
30 WARD	1250 22-FEB-1981	950
30 ALLEN	1600 20-FEB-1981	950

## 11.12 生成简单的预测

### 问题

以当前数据为基础，返回另外的行和列，用于表示未来活动。例如，查看下列结果集：

ID	ORDER_DATE	PROCESS_DATE
1	25-SEP-2005	27-SEP-2005
2	26-SEP-2005	28-SEP-2005
3	27-SEP-2005	29-SEP-2005

要求对于结果集中的每一行，都要返回 3 行（对于一个订单，原来有一行再外加两行），此外，还需要另外增加两列，用于存放对订单做进一步处理的日期。

从上面的结果集中可以看到，订单处理需要两天。对于这个例子，假定订单处理之后进行核对，最后一步是出货，订单处理完 1 天后进行核对，核对完后过 1 天就出货。显然希望能够将上面的结果集转换成如下结果集：

ID	ORDER_DATE	PROCESS_DATE	VERIFIED	SHIPPED
1	25-SEP-2005	27-SEP-2005		
1	25-SEP-2005	27-SEP-2005	28-SEP-2005	
1	25-SEP-2005	27-SEP-2005	28-SEP-2005	29-SEP-2005
2	26-SEP-2005	28-SEP-2005		
2	26-SEP-2005	28-SEP-2005	29-SEP-2005	
2	26-SEP-2005	28-SEP-2005	29-SEP-2005	30-SEP-2005
3	27-SEP-2005	29-SEP-2005		
3	27-SEP-2005	29-SEP-2005	30-SEP-2005	
3	27-SEP-2005	29-SEP-2005	30-SEP-2005	01-OCT-2005

### 解决方案

这里的关键是用笛卡儿积为每个订单生成两个额外行，然后，只要使用 CASE 表达式创建所需要的列值就可以了。

### DB2 和 SQL Server

使用递归 WITH 子句，生成笛卡儿积所需要的行。DB2 和 SQL Server 的解决方案相同，只是用于检索当前日期所使用的函数不同，DB2 使用 CURRENT\_DATE，而 SQL Server 使用 GETDATE。SQL Server 解决方案如下所示：

```

1 with nrows(n) as (
2   select 1 from t1 union all
3   select n+1 from nrows where n+1 <= 3
4 )
5 select id,
```

```

6         order_date,
7         process_date,
8         case when nrows.n >= 2
9             then process_date+1
10            else null
11        end as verified,
12        case when nrows.n = 3
13            then process_date+2
14            else null
15        end as shipped
16    from (
17    select nrows.n id,
18           getdate()+nrows.n as order_date,
19           getdate()+nrows.n+2 as process_date
20    from nrows
21    ) orders, nrows
22    order by 1

```

## Oracle

使用分层的 CONNECT BY 子句，生成笛卡儿积需要的 3 行信息。使用 WITH 子句，就可以重复使用由 CONNECT BY 返回的结果，而不必再次调用它：

```

1  with nrows as (
2    select level n
3    from dual
4    connect by level <= 3
5  )
6  select id,
7         order_date,
8         process_date,
9         case when nrows.n >= 2
10            then process_date+1
11            else null
12        end as verified,
13        case when nrows.n = 3
14            then process_date+2
15            else null
16        end as shipped
17    from (
18    select nrows.n id,
19           sysdate+nrows.n as order_date,
20           sysdate+nrows.n+2 as process_date
21    from nrows
22    ) orders, nrows

```

## PostgreSQL

可以采用很多不同的方法创建笛卡儿积。下面的解决方案使用了 PostgreSQL 函数 GENERATE\_SERIES：

```

1  select id,
2         order_date,
3         process_date,
4         case when gs.n >= 2
5             then process_date+1
6             else null
7         end as verified,
8         case when gs.n = 3
9             then process_date+2
10            else null
11        end as shipped
12    from (
13    select gs.id,

```

```

14      current_date+gs.id   as order_date,
15      current_date+gs.id+2 as process_date
16  from generate_series(1,3) gs (id)
17      ) orders,
18      generate_series(1,3)gs(n)

```

## MySQL

MySQL 并不支持能自动生成行的函数。

## 讨论

### DB2 和 SQL Server

“问题”一节中给出的结果集是由内联视图 ORDERS 返回的，如下所示：

```

with nrows(n) as (
select 1 from t1 union all
select n+1 from nrows where n+1 <= 3
)
select nrows.n id,
       getdate()+nrows.n   as order_date,
       getdate()+nrows.n+2 as process_date
  from nrows

```

ID	ORDER_DATE	PROCESS_DATE
1	25-SEP-2005	27-SEP-2005
2	26-SEP-2005	28-SEP-2005
3	27-SEP-2005	29-SEP-2005

上面的查询只是使用 WITH 子句产生 3 行，用于表示必须处理的订单。NROWS 会返回值 1、2 和 3，把这些数字与 GETDATE (DB2 是 CURRENT\_DATE) 的结果相加，表示订单的日期。因为“问题”一节中提出订单的处理需要花两天时间，上面的查询也要给 ORDER\_DATE 加两天（把由 NROWS 返回的值与 GETDATE 相加，然后再加 2 天）。

这样，就得到了基础结果集，下一步将创建笛卡儿积，其需求是为每个订单返回 3 行。使用 NROWS 可创建笛卡儿积，以便为每个订单返回 3 行：

```

with nrows(n) as (
select 1 from t1 union all
select n+1 from nrows where n+1 <= 3
)
select nrows.n,
       orders.*
  from (
select nrows.n id,
       getdate()+nrows.n   as order_date,
       getdate()+nrows.n+2 as process_date
  from nrows
  ) orders, nrows
 order by 2,1

```

N	ID	ORDER_DATE	PROCESS_DATE
1	1	25-SEP-2005	27-SEP-2005
2	1	25-SEP-2005	27-SEP-2005
3	1	25-SEP-2005	27-SEP-2005

```

1  2 26-SEP-2005 28-SEP-2005
2  2 26-SEP-2005 28-SEP-2005
3  2 26-SEP-2005 28-SEP-2005
1  3 27-SEP-2005 29-SEP-2005
2  3 27-SEP-2005 29-SEP-2005
3  3 27-SEP-2005 29-SEP-2005

```

至此，每个订单都包含 3 行信息，现在要做的是使用 CASE 表达式创建额外列值，用于表示核对和出货的状态。

每个订单中第 1 行的 VERIFIED 和 SHIPPED 都应该是 NULL 值，第 2 行的 SHIPPED 都应该是 NULL 值，第 3 行的所有列都应该是非 NULL 值。最终结果集如下所示：

```

with nrows(n) as (
select 1 from t1 union all
select n+1 from nrows where n+1 <= 3
)
select id,
       order_date,
       process_date,
       case when nrows.n >= 2
            then process_date+1
            else null
       end as verified,
       case when nrows.n = 3
            then process_date+2
            else null
       end as shipped
from (
select nrows.n id,
       getdate()+nrows.n as order_date,
       getdate()+nrows.n+2 as process_date
from nrows
) orders, nrows
order by 1

```

ID	ORDER_DATE	PROCESS_DATE	VERIFIED	SHIPPED
1	25-SEP-2005	27-SEP-2005		
1	25-SEP-2005	27-SEP-2005	28-SEP-2005	
1	25-SEP-2005	27-SEP-2005	28-SEP-2005	29-SEP-2005
2	26-SEP-2005	28-SEP-2005		
2	26-SEP-2005	28-SEP-2005	29-SEP-2005	
2	26-SEP-2005	28-SEP-2005	29-SEP-2005	30-SEP-2005
3	27-SEP-2005	29-SEP-2005		
3	27-SEP-2005	29-SEP-2005	30-SEP-2005	
3	27-SEP-2005	29-SEP-2005	30-SEP-2005	01-OCT-2005

最终结果集展示了完整的订单处理过程，即从接收订单开始到订单出货为止。

## Oracle

“问题”一节中给出的结果集是由内联视图 ORDERS 返回的，如下所示：

```

with nrows as (
select level n
from dual
connect by level <= 3
)
select nrows.n id,
       sysdate+nrows.n order_date,

```

```

        sysdate+nrows.n+2 process_date
from nrows

```

```

ID ORDER_DATE  PROCESS_DATE
-----
1 25-SEP-2005  27-SEP-2005
2 26-SEP-2005  28-SEP-2005
3 27-SEP-2005  29-SEP-2005

```

上面的查询只是使用CONNECT BY子句构成3行,用于表示必须处理的订单。使用WITH子句,可把CONNECT BY返回的行表示为NROWS.N。CONNECT BY会返回值1、2和3,把这些数字与SYSDATE相加,就表示订单的日期。因为“问题”一节中提出订单的处理需要花两天时间,上面的查询也要给ORDER\_DATE加两天(把由CONNECT BY(原文是GENERATE\_SERIES,有误,译者注)返回的值与SYSDATE相加,然后再加2天)。

这样,就得到了基础结果集,下一步将创建笛卡儿积,这要求是为每个订单返回3行。使用NROWS可创建笛卡儿积,以便为每个订单返回3行:

```

with nrows as (
select level n
  from dual
 connect by level <= 3
)
select nrows.n,
       orders.*
  from (
select nrows.n id,
       sysdate+nrows.n order_date,
       sysdate+nrows.n+2 process_date
  from nrows
       ) orders, nrows

```

```

N  ID ORDER_DATE  PROCESS_DATE
---
1  1 25-SEP-2005  27-SEP-2005
2  1 25-SEP-2005  27-SEP-2005
3  1 25-SEP-2005  27-SEP-2005
1  2 26-SEP-2005  28-SEP-2005
2  2 26-SEP-2005  28-SEP-2005
3  2 26-SEP-2005  28-SEP-2005
1  3 27-SEP-2005  29-SEP-2005
2  3 27-SEP-2005  29-SEP-2005
3  3 27-SEP-2005  29-SEP-2005

```

至此,每个订单都包含3行信息,现在要做的是使用CASE表达式创建2个额外列值,用于表示核对和出货的状态。

每个订单中第一行的VERIFIED和SHIPPED都应该是NULL值,第二行的SHIPPED都应该是NULL值,第三行的所有列都应该是非NULL值。最终结果集如下所示:

```

with nrows as (
select level n
  from dual
 connect by level <= 3
)
select id,

```

```

        order_date,
        process_date,
        case when nrows.n >= 2
            then process_date+1
            else null
        end as verified,
        case when nrows.n = 3
            then process_date+2
            else null
        end as shipped
    from (
    select nrows.n id,
        sysdate+nrows.n order_date,
        sysdate+nrows.n+2 process_date
    from nrows
    ) orders, nrows

```

ID	ORDER_DATE	PROCESS_DATE	VERIFIED	SHIPPED
1	25-SEP-2005	27-SEP-2005		
1	25-SEP-2005	27-SEP-2005	28-SEP-2005	
1	25-SEP-2005	27-SEP-2005	28-SEP-2005	29-SEP-2005
2	26-SEP-2005	28-SEP-2005		
2	26-SEP-2005	28-SEP-2005	29-SEP-2005	
2	26-SEP-2005	28-SEP-2005	29-SEP-2005	30-SEP-2005
3	27-SEP-2005	29-SEP-2005		
3	27-SEP-2005	29-SEP-2005	30-SEP-2005	
3	27-SEP-2005	29-SEP-2005	30-SEP-2005	01-OCT-2005

最终结果集展示了完整的订单处理过程，即从接收订单开始到订单出货为止。

## PostgreSQL

“问题”一节中给出的结果集是由内联视图 ORDERS 返回的，如下所示：

```

select gs.id,
    current_date+gs.id as order_date,
    current_date+gs.id+2 as process_date
from generate_series(1,3) gs (id)

```

ID	ORDER_DATE	PROCESS_DATE
1	25-SEP-2005	27-SEP-2005
2	26-SEP-2005	28-SEP-2005
3	27-SEP-2005	29-SEP-2005

上面的查询只是使用 GENERATE\_SERIES 函数构成 3 行，用于表示必须要处理的订单。GENERATE\_SERIES 函数将返回值 1、2 和 3，把这些数字与 CURRENT\_DATE 相加，就表示订单的日期。因为“问题”一节中提出订单的处理需要花两天时间，上面的查询也要给 ORDER\_DATE 加两天（把由 GENERATE\_SERIES 返回的值与 CURRENT\_DATE 相加，然后再加 2 天）。

这样，就得到了基础结果集，下一步将创建笛卡儿积，其需求是为每个订单返回 3 行。使用 GENERATE\_SERIES 函数可创建笛卡儿积，以便为每个订单返回 3 行：

```

select gs.n,
    orders.*
from (
select gs.id,

```

```

        current_date+gs.id    as order_date,
        current_date+gs.id+2 as process_date
    from generate_series(1,3) gs (id)
    ) orders,
    generate_series(1,3)gs(n)

```

N	ID	ORDER_DATE	PROCESS_DATE
1	1	25-SEP-2005	27-SEP-2005
2	1	25-SEP-2005	27-SEP-2005
3	1	25-SEP-2005	27-SEP-2005
1	2	26-SEP-2005	28-SEP-2005
2	2	26-SEP-2005	28-SEP-2005
3	2	26-SEP-2005	28-SEP-2005
1	3	27-SEP-2005	29-SEP-2005
2	3	27-SEP-2005	29-SEP-2005
3	3	27-SEP-2005	29-SEP-2005

至此，每个订单都包含 3 行信息，现在要做的是使用 CASE 表达式创建额外列值，用于表示核对和出货的状态。

每个订单中第 1 行的 VERIFIED 和 SHIPPED 都应该是 NULL 值，第 2 行的 SHIPPED 都应该是 NULL 值，第 3 行的所有列都应该是非 NULL 值。最终结果集如下所示：

```

select id,
       order_date,
       process_date,
       case when gs.n >= 2
            then process_date+1
            else null
       end as verified,
       case when gs.n = 3
            then process_date+2
            else null
       end as shipped
    from (
select gs.id,
       current_date+gs.id    as order_date,
       current_date+gs.id+2 as process_date
    from generate_series(1,3) gs(id)
    ) orders,
    generate_series(1,3)gs(n)

```

ID	ORDER_DATE	PROCESS_DATE	VERIFIED	SHIPPED
1	25-SEP-2005	27-SEP-2005		
1	25-SEP-2005	27-SEP-2005	28-SEP-2005	
1	25-SEP-2005	27-SEP-2005	28-SEP-2005	29-SEP-2005
2	26-SEP-2005	28-SEP-2005		
2	26-SEP-2005	28-SEP-2005	29-SEP-2005	
2	26-SEP-2005	28-SEP-2005	29-SEP-2005	30-SEP-2005
3	27-SEP-2005	29-SEP-2005		
3	27-SEP-2005	29-SEP-2005	30-SEP-2005	
3	27-SEP-2005	29-SEP-2005	30-SEP-2005	01-OCT-2005

最终结果集展示了完整的订单处理过程，即从接收订单开始到订单出货为止。

## 第 12 章

# 报表和数据仓库运算

本章将介绍几种查询，它们对创建报表非常有帮助。其代表性的内容有针对性报表格式的一些因素以及不同层的聚集。本章的另一个重点是转置变换结果集、把行转换为列。转置变换对解决许多问题是一种非常有用的技巧。经转置变换会使人感觉更舒服，除本章所介绍的之外，无疑还有其他用途。

### 12.1 将结果集转置为一行

#### 问题

希望将几个行组中的数据转换成几行中的列，每个原来的行组转换成一行。例如，下面的结果集显示了每个部门中员工的数目：

DEPTNO	CNT
10	3
20	5
30	6

希望重新设置输出格式，使其结果集看起来如下：

DEPTNO_10	DEPTNO_20	DEPTNO_30
3	5	6

#### 解决方案

使用 CASE 表达式和聚集函数 SUM，变换结果集：

```
1 select sum(case when deptno=10 then 1 else 0 end) as deptno_10,
2        sum(case when deptno=20 then 1 else 0 end) as deptno_20,
3        sum(case when deptno=30 then 1 else 0 end) as deptno_30
4 from emp
```

#### 讨论

这个例子引入了转置变换。其思想非常简单：对于由非转置查询返回的所有行，使用



CASE表达式把各行数据分成列。由于这个特定问题是计算每个部门的员工数，所以可使用聚集函数 SUM 计算每个 DEPTNO 出现的次数。如果不能理解这种做法，则可以执行带有聚集函数 SUM 的查询，而且为提高可读性，列出了 DEPTNO：

```
select deptno,
       case when deptno=10 then 1 else 0 end as deptno_10,
       case when deptno=20 then 1 else 0 end as deptno_20,
       case when deptno=30 then 1 else 0 end as deptno_30
from emp
order by 1
```

DEPTNO	DEPTNO_10	DEPTNO_20	DEPTNO_30
10	1	0	0
10	1	0	0
10	1	0	0
20	0	1	0
20	0	1	0
20	0	1	0
20	0	1	0
20	0	1	0
30	0	0	1
30	0	0	1
30	0	0	1
30	0	0	1
30	0	0	1
30	0	0	1

可以把每个CASE表达式看作一个标志，用以确定一个行属于哪个DEPTNO。至此，“行到列”的转换已经完成了。下一步，只需分别把 DEPTNO\_10、DEPTNO\_20 和 DEPTNO\_30 返回的值相加，然后再按 DEPTNO 分组。下面列出了结果：

```
select deptno,
       sum(case when deptno=10 then 1 else 0 end) as deptno_10,
       sum(case when deptno=20 then 1 else 0 end) as deptno_20,
       sum(case when deptno=30 then 1 else 0 end) as deptno_30
from emp
group by deptno
```

DEPTNO	DEPTNO_10	DEPTNO_20	DEPTNO_30
10	3	0	0
20	0	5	0
30	0	0	6

观察一下这个结果集就会看到：从逻辑上讲该输出很容易理解；例如，在 DEPTNO\_10 中，DEPTNO 10 有 3 名员工，而其他部门的员工为 0。由于目标是只返回一行，最后一步是丢掉 DEPTNO 和 GROUP BY，并简单地将 CASE 表达式的值加起来：

```
select sum(case when deptno=10 then 1 else 0 end) as deptno_10,
       sum(case when deptno=20 then 1 else 0 end) as deptno_20,
       sum(case when deptno=30 then 1 else 0 end) as deptno_30
from emp
```

DEPTNO_10	DEPTNO_20	DEPTNO_30
3	5	6

对于这种类型的问题，还可采用下列方法进行处理：

```

select max(case when deptno=10 then empcount else null end) as deptno_10
       max(case when deptno=20 then empcount else null end) as deptno_20,
       max(case when deptno=30 then empcount else null end) as deptno_30
from (
select deptno, count(*) as empcount
from emp
group by deptno
) x

```

在这种方法中，使用内联视图生成每个部门的员工数。主查询中的 CASE 表达式把行转换为列，可得到下列结果集：

DEPTNO_10	DEPTNO_20	DEPTNO_30
3	NULL	NULL
NULL	5	NULL
NULL	NULL	6

然后，使用 MAX 函数把这些列转换为一行：

DEPTNO_10	DEPTNO_20	DEPTNO_30
3	5	6

## 12.2 把结果集转置为多行

### 问题

要把行转换为列，根据原表给定列的每个值创建一个列。然而，与上一节不同的是，这里需要输出多个行。

例如，返回每个员工及他们的职位 (JOB)，目前的查询返回如下结果集：

JOB	ENAME
ANALYST	SCOTT
ANALYST	FORD
CLERK	SMITH
CLERK	ADAMS
CLERK	MILLER
CLERK	JAMES
MANAGER	JONES
MANAGER	CLARK
MANAGER	BLAKE
PRESIDENT	KING
SALESMAN	ALLEN
SALESMAN	MARTIN
SALESMAN	TURNER
SALESMAN	WARD

希望重新设置结果集的格式，使每个职位使用一列：

CLERKS	ANALYSTS	MGRS	PREZ	SALES
MILLER	FORD	CLARK	KING	TURNER
JAMES	SCOTT	BLAKE		MARTIN
ADAMS		JONES		WARD
SMITH				ALLEN

## 解决方案

与本章第一节不同的是，本节的结果集包含多行。前一节介绍的技巧解决不了本节的问题，因为每个JOB的MAX(ENAME)都会有返回值，这将导致一个JOB只有一个ENAME（即，像上一节那样，总共只返回一行）。要解决这个问题，必须使每个JOB/ENAME组合唯一，然后，在使用聚集函数去除NULL时，不会丢失ENAME。

### DB2、Oracle 和 SQL Server

使用窗口函数ROW\_NUMBER OVER，使每个JOB/ENAME组合唯一。使用CASE表达式和聚集函数MAX对结果集进行转置变换，同时按窗口函数的返回值分组：

```
1 select max(case when job='CLERK'
2               then ename else null end) as clerks,
3        max(case when job='ANALYST'
4               then ename else null end) as analysts,
5        max(case when job='MANAGER'
6               then ename else null end) as mgrs,
7        max(case when job='PRESIDENT'
8               then ename else null end) as prez,
9        max(case when job='SALESMAN'
10              then ename else null end) as sales
11      from (
12 select job,
13        ename,
14        row_number()over(partition by job order by ename) rn
15      from emp
16       ) x
17     group by rn
```

### PostgreSQL 和 MySQL

使用标量子查询，按EMPNO给每个员工分等级。使用CASE表达式和聚集函数MAX对结果集进行转置变换，同时按子查询的返回值分组：

```
1 select max(case when job='CLERK'
2               then ename else null end) as clerks,
3        max(case when job='ANALYST'
4               then ename else null end) as analysts,
5        max(case when job='MANAGER'
6               then ename else null end) as mgrs,
7        max(case when job='PRESIDENT'
8               then ename else null end) as prez,
9        max(case when job='SALESMAN'
10              then ename else null end) as sales
11      from (
12 select e.job,
13        e.ename,
14        (select count(*) from emp d
15         where e.job=d.job and e.empno < d.empno) as rnk
16      from emp e
17       ) x
18     group by rnk
```

## 讨论

### DB2、Oracle 和 SQL Server

首先，使用窗口函数 ROW\_NUMBER OVER，使每个 JOB/ENAME 组合唯一：

```
select job,
       ename,
       row_number() over(partition by job order by ename) rn
from emp
```

JOB	ENAME	RN
ANALYST	FORD	1
ANALYST	SCOTT	2
CLERK	ADAMS	1
CLERK	JAMES	2
CLERK	MILLER	3
CLERK	SMITH	4
MANAGER	BLAKE	1
MANAGER	CLARK	2
MANAGER	JONES	3
PRESIDENT	KING	1
SALESMAN	ALLEN	1
SALESMAN	MARTIN	2
SALESMAN	TURNER	3
SALESMAN	WARD	4

对于一个给定职位，每个 ENAME 都有唯一的“行号”，这就可以避免可能有两个员工名字和职位都相同而带来的问题。这样做的目的是能够按行号 (RN) 分组，在使用了 MAX 时不会遗漏结果集中的任何员工。这是解决本问题最重要的步骤，如果没有这一步，外层查询的聚集运算会丢掉一些有用行。如果不使用 ROW\_NUMBER OVER，而采用第一节介绍的技巧，就会产生以下结果集：

```
select max(case when job='CLERK'
               then ename else null end) as clerks,
       max(case when job='ANALYST'
               then ename else null end) as analysts,
       max(case when job='MANAGER'
               then ename else null end) as mgrs,
       max(case when job='PRESIDENT'
               then ename else null end) as prez,
       max(case when job='SALESMAN'
               then ename else null end) as sales
from emp
```

CLERKS	ANALYSTS	MGRS	PREZ	SALES
SMITH	SCOTT	JONES	KING	WARD

很遗憾，每个 JOB 只返回一行：即 ENAME 最大的员工。当转置结果集时，MIN 或 MAX 的作用应该是从结果集中去掉 NULL，而不是限制返回的 ENAME。随着后面解释的深入，这种方式的机理会越来越清晰。

下一步，使用 CASE 表达式，把 ENAME 组成相应的列 (JOB)：

```
select rn,
       case when job='CLERK'
            then ename else null end as clerks,
```

```

        case when job='ANALYST'
            then ename else null end as analysts,
        case when job='MANAGER'
            then ename else null end as mgrs,
        case when job='PRESIDENT'
            then ename else null end as prez,
        case when job='SALESMAN'
            then ename else null end as sales
    from (
select job,
       ename,
       row_number()over(partition by job order by ename) rn
    from emp
    ) x

```

RN	CLERKS	ANALYSTS	MGRS	PREZ	SALES
1		FORD			
2		SCOTT			
1	ADAMS				
2	JAMES				
3	MILLER				
4	SMITH				
1			BLAKE		
2			CLARK		
3			JONES		
1				KING	
1					ALLEN
2					MARTIN
3					TURNER
4					WARD

至此，所有行都变换为列。最后一步，剔除NULL，使结果集可读性更好。要剔除NULL，可使用聚集函数MAX，并按RN分组（因为每组仅需对一个值作聚集，也可以使用MIN函数，MAX只是随意选择的）。每个RN/JOB/ENAME组合都只有一个值。按RN分组，并把CASE表达式嵌入MAX调用内，就能够确保每个MAX调用都会从该组内选择一个名字，而不会选择NULL值：

```

select max(case when job='CLERK'
                then ename else null end) as clerks,
       max(case when job='ANALYST'
                then ename else null end) as analysts,
       max(case when job='MANAGER'
                then ename else null end) as mgrs,
       max(case when job='PRESIDENT'
                then ename else null end) as prez,
       max(case when job='SALESMAN'
                then ename else null end) as sales
    from (
select job,
       ename,
       row_number()over(partition by job order by ename) rn
    from emp
    ) x
group by rn

```

	CLERKS	ANALYSTS	MGRS	PREZ	SALES
MILLER	FORD		CLARK	KING	TURNER
JAMES	SCOTT		BLAKE		MARTIN
ADAMS			JONES		WARD
SMITH					ALLEN

使用 ROW\_NUMBER OVER 创建唯一行组合，采用这种技巧对设置查询结果的格式非常有用。下面的查询创建了一个稀疏报表，它按照 DEPTNO 和 JOB 显示员工：

```
select deptno dno, job,
       max(case when deptno=10
                 then ename else null end) as d10,
       max(case when deptno=20
                 then ename else null end) as d20,
       max(case when deptno=30
                 then ename else null end) as d30,
       max(case when job='CLERK'
                 then ename else null end) as clerks,
       max(case when job='ANALYST'
                 then ename else null end) as anals,
       max(case when job='MANAGER'
                 then ename else null end) as mgrs,
       max(case when job='PRESIDENT'
                 then ename else null end) as prez,
       max(case when job='SALESMAN'
                 then ename else null end) as sales
from (
select deptno,
       job,
       ename,
       row_number()over(partition by job order by ename) rn_job,
       row_number()over(partition by job order by ename) rn_deptno
from emp
) x
group by deptno, job, rn_deptno, rn_job
order by 1
```

DNO	JOB	D10	D20	D30	CLERKS	ANALS	MGRS	PREZ	SALES
10	CLERK	MILLER			MILLER				
10	MANAGER	CLARK					CLARK		
10	PRESIDENT	KING						KING	
20	ANALYST		FORD			FORD			
20	ANALYST		SCOTT			SCOTT			
20	CLERK		ADAMS		ADAMS				
20	CLERK		SMITH		SMITH				
20	MANAGER		JONES				JONES		
30	CLERK			JAMES	JAMES				
30	MANAGER			BLAKE			BLAKE		
30	SALESMAN			ALLEN					ALLEN
30	SALESMAN			MARTIN					MARTIN
30	SALESMAN			TURNER					TURNER
30	SALESMAN			WARD					WARD

只要修改一下分组条件（因而也要修改 SELECT 列表中的非聚集项），就可以产生不同格式的报表。为了理解 GROUP BY 语句中的条件对格式有何影响，需要反复改来改去，这样做还是很值得的。

## PostgreSQL 和 MySQL

一旦创建了 JOB/ENAME 的唯一组合，这两种 RDBMS 的技巧与其他平台相同。第一步，使用标量子查询为每个 JOB/ENAME 组合提供“行号”或“等级”：

```
select e.job,
       e.ename,
       (select count(*) from emp d
        where e.job=d.job and e.empno < d.empno) as rnk
```

```

from emp e

```

JOB	ENAME	RNK
CLERK	SMITH	3
SALESMAN	ALLEN	3
SALESMAN	WARD	2
MANAGER	JONES	2
SALESMAN	MARTIN	1
MANAGER	BLAKE	1
MANAGER	CLARK	0
ANALYST	SCOTT	1
PRESIDENT	KING	0
SALESMAN	TURNER	0
CLERK	ADAMS	2
CLERK	JAMES	1
ANALYST	FORD	0
CLERK	MILLER	0

给每个JOB/ENAME组合分配一个唯一的“等级”，就使每行都唯一。即使同名的员工拥有同样的职位，也不会有两个员工职位的等级都相同。这是解决本问题最重要的步骤。如果没有这一步，外层查询中的聚集会去除有用的行。如果给每个JOB/ENAME组合加上等级，而采用第一节中介绍的技巧，就会产生如下结果集：

```

select max(case when job='CLERK'
                then ename else null end) as clerks,
       max(case when job='ANALYST'
                then ename else null end) as analysts,
       max(case when job='MANAGER'
                then ename else null end) as mgrs,
       max(case when job='PRESIDENT'
                then ename else null end) as prez,
       max(case when job='SALESMAN'
                then ename else null end) as sales
from emp

```

CLERKS	ANALYSTS	MGRS	PREZ	SALES
SMITH	SCOTT	JONES	KING	WARD

很遗憾，每个JOB只返回一行：即ENAME值最大的员工。当转置结果集时，MIN或MAX的作用应该是从结果集中去掉NULL，而不是制约返回的ENAME。

现在，已经看到了使用等级的作用，可以进入下一步了。接下来。使用CASE表达式，把ENAME组成相应的列（JOB）：

```

select rnk,
       case when job='CLERK'
            then ename else null end as clerks,
       case when job='ANALYST'
            then ename else null end as analysts,
       case when job='MANAGER'
            then ename else null end as mgrs,
       case when job='PRESIDENT'
            then ename else null end as prez,
       case when job='SALESMAN'
            then ename else null end as sales
from (
select e.job,
       e.ename,
       (select count(*) from emp d

```

```

        where e.job=d.job and e.empno < d.empno) as rnk
from emp e
) x

```

RNK	CLERKS	ANALYSTS	MGRS	PREZ	SALES
3	SMITH				
3					ALLEN
2					WARD
2			JONES		
1					MARTIN
1			BLAKE		
0			CLARK		
1		SCOTT			
0				KING	
0					TURNER
2	ADAMS				
1	JAMES				
0		FORD			
0	MILLER				

至此,所有行都变换为列。最后一步,剔除NULL,使结果集可读性更好。要剔除NULL,可使用聚集函数MAX,并按RNK分组(因为每组仅需对一个值作聚集,也可以使用MIN函数,MAX只是随意选择的)。每个RN/JOB/ENAME组合都只有一个值。聚集函数的作用只是去除NULL值:

```

select max(case when job='CLERK'
                then ename else null end) as clerks,
       max(case when job='ANALYST'
                then ename else null end) as analysts,
       max(case when job='MANAGER'
                then ename else null end) as mgrs,
       max(case when job='PRESIDENT'
                then ename else null end) as prez,
       max(case when job='SALESMAN'
                then ename else null end) as sales
from (
select e.job,
       e.ename,
       (select count(*) from emp d
        where e.job=d.job and e.empno < d.empno) as rnk
from emp e
) x
group by rnk

```

	CLERKS	ANALYSTS	MGRS	PREZ	SALES
MILLER	FORD	CLARK	KING	TURNER	
JAMES	SCOTT	BLAKE		MARTIN	
ADAMS		JONES		WARD	
SMITH				ALLEN	

## 12.3 反向转置结果集

### 问题

把列转换为行。请看下列结果集:

DEPTNO_10	DEPTNO_20	DEPTNO_30
3	5	6



要把它转换为：

DEPTNO	COUNTS_BY_DEPT
10	3
20	5
30	6

## 解决方案

检验想要的结果集，很容易明白，对表 EMP 执行简单的 COUNT 和 GROUP BY，就能产生想要的结果。尽管如此，这里的目的是假设数据按行存储，也许数据是以多列存储、非规范化和聚集过的值。

为把列转换为行，要使用笛卡儿积。需要提前知道要把多少列转换为行，因为用于创建笛卡儿积的表达式至少得拥有要转换的列数。

本节的解决方案没有创建非规范化的数据表，而是使用本章第一节介绍的解决方案创建一个“宽”结果集。完整的解决方案如下所示：

```

1 select dept.deptno,
2        case dept.deptno
3            when 10 then emp_cnts.deptno_10
4            when 20 then emp_cnts.deptno_20
5            when 30 then emp_cnts.deptno_30
6        end as counts_by_dept
7   from (
8 select sum(case when deptno=10 then 1 else 0 end) as deptno_10,
9        sum(case when deptno=20 then 1 else 0 end) as deptno_20,
10       sum(case when deptno=30 then 1 else 0 end) as deptno_30
11   from emp
12      ) emp_cnts,
13      (select deptno from dept where deptno <= 30) dept

```

## 讨论

内联视图 EMP\_CNTS 表示非规范化的视图，要转换为行的“宽”结果集如下所示：

```

select sum(case when deptno=10 then 1 else 0 end) as deptno_10,
       sum(case when deptno=20 then 1 else 0 end) as deptno_20,
       sum(case when deptno=30 then 1 else 0 end) as deptno_30
  from emp

```

DEPTNO_10	DEPTNO_20	DEPTNO_30
3	5	6

因为只有3列，所以会创建3行。首先，在内联视图 EMP\_CNTS 和某些表表达式（至少包含3行）之间创建笛卡儿积。在下面的代码中，使用了表 DEPT 创建笛卡儿积；DEPT 包含4行：

```

select dept.deptno,
       emp_cnts.deptno_10,
       emp_cnts.deptno_20,
       emp_cnts.deptno_30
  from (

```

```

select sum(case when deptno=10 then 1 else 0 end) as deptno_10,
       sum(case when deptno=20 then 1 else 0 end) as deptno_20,
       sum(case when deptno=30 then 1 else 0 end) as deptno_30
  from emp
       ) emp_cnts,
       (select deptno from dept where deptno <= 30) dept

```

DEPTNO	DEPTNO_10	DEPTNO_20	DEPTNO_30
10	3	5	6
20	3	5	6
30	3	5	6

笛卡儿积可以为内联视图 EMP\_CNTS 的每一列返回一行。由于最终结果集应该仅包含 DEPTNO 和员工数，所以可使用 CASE 表达式把 3 列转换为一列：

```

select dept.deptno,
       case dept.deptno
         when 10 then emp_cnts.deptno_10
         when 20 then emp_cnts.deptno_20
         when 30 then emp_cnts.deptno_30
       end as counts_by_dept
  from (
select sum(case when deptno=10 then 1 else 0 end) as deptno_10,
       sum(case when deptno=20 then 1 else 0 end) as deptno_20,
       sum(case when deptno=30 then 1 else 0 end) as deptno_30
  from emp
       ) emp_cnts,
       (select deptno from dept where deptno <= 30) dept

```

DEPTNO	COUNTS_BY_DEPT
10	3
20	5
30	6

## 12.4 将结果集反向转置为一列

### 问题

把查询中返回的所有列转换为 1 列。例如，返回 DEPTNO 10 中所有员工的 ENAME、JOB 和 SAL，而且要把这 3 个值放到一列中。为每个员工返回 3 行信息，而且在两个员工之间加一个空白行。希望返回的结果集如下：

```

EMPS
-----
CLARK
MANAGER
2450

KING
PRESIDENT
5000

MILLER
CLERK
1300

```

解决方案

关键是用笛卡儿积为每个员工返回 4 行。这样就可以为每列产生一行，而且在两个员工之间加一个空白行。

DB2、Oracle 和 SQL Server

使用窗口函数 ROW\_NUMBER OVER，基于 EMPNO (1~4)给每行分等级。然后，使用 CASE 表达式把 3 列转换为 1 列：

```
1  select case rn
2           when 1 then ename
3           when 2 then job
4           when 3 then cast(sal as char(4))
5           end emps
6  from (
7  select e.ename,e.job,e.sal,
8         row_number()over(partition by e.empno
9                           order by e.empno) rn
10 from emp e,
11      (select *
12       from emp where job='CLERK') four_rows
13 where e.deptno=10
14      ) x
```

PostgreSQL 和 MySQL

本节的主要意图是用窗口函数给行分等级，然后在进行转置时派用场。到编写本书时，PostgreSQL 和 MySQL 都不支持窗口函数。

讨论

DB2、Oracle 和 SQL Server

首先，使用窗口函数 ROW\_NUMBER OVER，为 DEPTNO 10 中的每个员工创建等级：

```
select e.ename,e.job,e.sal,
       row_number()over(partition by e.empno
                         order by e.empno) rn
from emp e
where e.deptno=10
```

ENAME	JOB	SAL	RN
CLARK	MANAGER	2450	1
KING	PRESIDENT	5000	1
MILLER	CLERK	1300	1

此时，等级没有多大的意义。由于是按照 EMPNO 进行分区的，所以对于 DEPTNO 10 中的 3 行，其等级都为 1。一旦添加了笛卡儿积，等级就体现出来了，从下面的结果中可以看出：

```
select e.ename,e.job,e.sal,
       row_number()over(partition by e.empno
                         order by e.empno) rn
from emp e,
```

```
(select *
  from emp where job='CLERK') four_rows
where e.deptno=10
```

ENAME	JOB	SAL	RN
CLARK	MANAGER	2450	1
CLARK	MANAGER	2450	2
CLARK	MANAGER	2450	3
CLARK	MANAGER	2450	4
KING	PRESIDENT	5000	1
KING	PRESIDENT	5000	2
KING	PRESIDENT	5000	3
KING	PRESIDENT	5000	4
MILLER	CLERK	1300	1
MILLER	CLERK	1300	2
MILLER	CLERK	1300	3
MILLER	CLERK	1300	4

现在停一下，先理解两个关键点：

- 每个员工的 RN 不再为 1，变为从 1 至 4 的重复序列，其原因是：在计算 FROM 和 WHERE 子句之后窗口函数起作用了。由于按 EMPNO 分区，当遇到新员工时就把 RN 值恢复为 1。
- 内联视图 FOUR\_ROWS 非常简单，就是一句返回 4 行的 SQL 语句，为每个列 (ENAME、JOB、SAL) 返回一行，另加一行作为空白。

此时，艰苦工作已经完成，剩下的就是使用 CASE 表达式把每个员工的 ENAME、JOB 和 SAL 放到一个列中（为了满足 CASE 的要求，需要把 SAL 转换为字符串）：

```
select case rn
  when 1 then ename
  when 2 then job
  when 3 then cast(sal as char(4))
end emps
  from (
select e.ename,e.job,e.sal,
  row_number()over(partition by e.empno
                    order by e.empno) rn
  from emp e,
  (select *
   from emp where job='CLERK') four_rows
 where e.deptno=10
  ) x
```

```
EMPS
-----
CLARK
MANAGER
2450
KING
PRESIDENT
5000
MILLER
CLERK
1300
```

## 12.5 抑制结果集中的重复值

### 问题

生成报表时如果两行的同一列包含相同值，希望这个值仅显示一次。例如，要从表 EMP 中找出 DEPTNO 和 ENAME，按 DEPTNO 给所有行分组，而且 DEPTNO 仅显示一次。希望返回如下的结果集：

DEPTNO	ENAME
10	CLARK
	KING
	MILLER
20	SMITH
	ADAMS
	FORD
	SCOTT
	JONES
30	ALLEN
	BLAKE
	MARTIN
	JAMES
	TURNER
	WARD

### 解决方案

这是非常简单的格式设置问题，使用 Oracle 提供的窗口函数 LAG OVER 能够很容易解决这个问题。也可以使用其他方法（如标量子查询）及其他窗口函数（对于非 Oracle 平台，必须使用这些函数），但这里 LAG OVER 是最方便、最合适的。

### DB2 和 SQL Server

可以使用窗口函数 MIN OVER，找出每个 DEPTNO 的最小 EMPNO；然后，使用 CASE 表达式对 EMPNO 为其他值的行进行“清空”：

```
1 select case when empno=min_empno
2             then deptno else null
3             end deptno,
4             ename
5   from (
6 select deptno,
7        min(empno)over(partition by deptno) min_empno,
8        empno,
9        ename
10  from emp
11  ) x
```

### Oracle

使用窗口函数 LAG OVER，访问当前行的前几行，以便为每个分区找到第一个 DEPTNO：

```
1 select to_number(
2         decode(lag(deptno)over(order by deptno),
3         deptno,null,deptno)
4       ) deptno, ename
5   from emp
```

## PostgreSQL 和 MySQL

本节的主要意图是用窗口函数访问当前行附近的行。编写本书时, PostgreSQL 和 MySQL 都不支持窗口函数。

## 讨论

### DB2 和 SQL Server

首先, 使用窗口函数 MIN OVER 找到每个 DEPTNO 中最小的 EMPNO:

```
select deptno,
       min(empno)over(partition by deptno) min_empno,
       empno,
       ename
from emp
```

DEPTNO	MIN_EMPNO	EMPNO	ENAME
10	7782	7782	CLARK
10	7782	7839	KING
10	7782	7934	MILLER
20	7369	7369	SMITH
20	7369	7876	ADAMS
20	7369	7902	FORD
20	7369	7788	SCOTT
20	7369	7566	JONES
30	7499	7499	ALLEN
30	7499	7698	BLAKE
30	7499	7654	MARTIN
30	7499	7900	JAMES
30	7499	7844	TURNER
30	7499	7521	WARD

下一步, 也就是最后一步, 使用 CASE 表达式抑制 DEPTNO 的重复显示: 如果某个员工的 EMPNO 与 MIN\_EMPNO 相匹配, 则返回 DEPTNO, 否则返回 NULL:

```
select case when empno=min_empno
           then deptno else null
       end deptno,
       ename
from (
select deptno,
       min(empno)over(partition by deptno) min_empno,
       empno,
       ename
from emp
) x
```

DEPTNO	ENAME
10	CLARK
	KING
	MILLER
20	SMITH
	ADAMS
	FORD
	SCOTT
	JONES
30	ALLEN
	BLAKE
	MARTIN

JAMES  
TURNER  
WARD

Oracle

首先，使用窗口函数 LAG OVER，为每行返回它的前一个 DEPTNO：

```
select lag(deptno)over(order by deptno) lag_deptno,
       deptno,
       ename
from emp
```

LAG_DEPTNO	DEPTNO	ENAME
	10	CLARK
10	10	KING
10	10	MILLER
10	20	SMITH
20	20	ADAMS
20	20	FORD
20	20	SCOTT
20	20	JONES
20	30	ALLEN
30	30	BLAKE
30	30	MARTIN
30	30	JAMES
30	30	TURNER
30	30	WARD

观察上面的结果集就会明白，对于 DEPTNO 与 LAG\_DEPTNO 相匹配的行，需要把 DEPTNO 设置为 NULL。使用 DECODE 可完成此功能（使用 TO\_NUMBER 是为了把 DEPTNO 转换为数值）：

```
select to_number(
       decode(lag(deptno)over(order by deptno),
              deptno,null,deptno)
       ) deptno, ename
from emp
```

DEPTNO	ENAME
	CLARK
	KING
	MILLER
20	SMITH
	ADAMS
	FORD
	SCOTT
	JONES
30	ALLEN
	BLAKE
	MARTIN
	JAMES
	TURNER
	WARD

12.6 转置结果集以利于跨行计算

问题

对来自多个行的数据进行计算。为便于计算，可以把这些行转置到列中，这样所有需要的值都包含于 1 行中。

本书的示例数据中，DEPTNO 20是总工资最高的部门，执行下列查询可以确认这一点：

```
select deptno, sum(sal) as sal
  from emp
 group by deptno
```

DEPTNO	SAL
10	8750
20	10875
30	9400

现在要计算 DEPTNO 20 和 DEPTNO 10 之间以及 DEPTNO 20 和 DEPTNO 30 之间的总工资之差。

## 解决方案

使用聚集函数 SUM 及 CASE 表达式转换总和，然后，在 SELECT 列表中编写表达式：

```
1 select d20_sal - d10_sal as d20_10_diff,
2        d20_sal - d30_sal as d20_30_diff
3   from (
4 select sum(case when deptno=10 then sal end) as d10_sal,
5        sum(case when deptno=20 then sal end) as d20_sal,
6        sum(case when deptno=30 then sal end) as d30_sal
7   from emp
8   ) totals_by_dept
```

## 讨论

首先使用 CASE 表达式，把每个 DEPTNO 的工资从行转置到列：

```
select case when deptno=10 then sal end as d10_sal,
       case when deptno=20 then sal end as d20_sal,
       case when deptno=30 then sal end as d30_sal
  from emp
```

D10_SAL	D20_SAL	D30_SAL
	800	1600
		1250
	2975	1250
		2850
2450	3000	
5000		
		1500
	1100	
		950
	3000	
1300		

下一步，对每个 CASE 表达式使用聚集函数 SUM，以便计算每个 DEPTNO 中所有工资的总和：

```
select sum(case when deptno=10 then sal end) as d10_sal,
       sum(case when deptno=20 then sal end) as d20_sal,
       sum(case when deptno=30 then sal end) as d30_sal
  from emp
```



D10_SAL	D20_SAL	D30_SAL
8750	10875	9400

最后一步，把上面的 SQL 包入内联视图，并进行减法计算。

## 12.7 创建固定大小的数据桶

### 问题

把数据编组成大小均匀的桶，每桶都包含预定的元素数。桶的总数可能不能确定，但要确保每桶都包含 5 个元素。例如，依据 EMPNO 值把表 EMP 中的员工分成 5 个一组，下面给出了结果：

GRP	EMPNO	ENAME
1	7369	SMITH
1	7499	ALLEN
1	7521	WARD
1	7566	JONES
1	7654	MARTIN
2	7698	BLAKE
2	7782	CLARK
2	7788	SCOTT
2	7839	KING
2	7844	TURNER
3	7876	ADAMS
3	7900	JAMES
3	7902	FORD
3	7934	MILLER

### 解决方案

如果 RDBMS 提供了给行分等级的函数，则对问题的解决方案就非常简单了。一旦给行分了等级，创建包含 5 个元素的桶就是很简单的除法操作，对商向上取整即可。

#### DB2、Oracle 和 SQL Server

使用窗口函数 ROW\_NUMBER OVER，按 EMPNO 给每个员工分等级。然后，除以 5，就创建了组（SQL Server 用户使用 CEILING，而不是 CEIL）：

```
1 select ceil(row_number()over(order by empno)/5.0) grp,
2      empno,
3      ename
4 from emp
```

#### PostgreSQL 和 MySQL

使用标量子查询，给每个 EMPNO 分等级。然后，除以 5，创建组：

```
1 select ceil(rnk/5.0) as grp,
2      empno, ename
3 from (
4 select e.empno, e.ename,
5      (select count(*) from emp d
6       where e.empno < d.empno)+1 as rnk
7 from emp e
```

```

8      ) x
9      order by grp

```

## 讨论

### DB2、Oracle 和 SQL Server

窗口函数 ROW\_NUMBER OVER 用于给按 EMPNO 排序的每一行分配一个等级或“行号”：

```

select row_number()over(order by empno) rn,
       empno,
       ename
from emp

```

RN	EMPNO	ENAME
1	7369	SMITH
2	7499	ALLEN
3	7521	WARD
4	7566	JONES
5	7654	MARTIN
6	7698	BLAKE
7	7782	CLARK
8	7788	SCOTT
9	7839	KING
10	7844	TURNER
11	7876	ADAMS
12	7900	JAMES
13	7902	FORD
14	7934	MILLER

下一步，在 ROW\_NUMBER OVER 除以 5 之后，使用函数 CEIL（或 CEILING）。从逻辑上讲，除以 5 就是把行编成 5 个一组的桶，即小于等于 1 的五个值、大于 1 但小于等于 2 的五个值，另一组（包括最后 4 行，这是由于表 EMP 中的行数是 14，不是 5 的倍数）包含大于 2 但小于等于 3 的值。

CEIL 函数返回一个比传递给它的值大的最小整数，这就创建了整数组。下面给出了做除法和使用 CEIL 的结果。可以按从左到右的顺序看，从 RN 到 DIVISION 再到 GRP：

```

select row_number()over(order by empno) rn,
       row_number()over(order by empno)/5.0 division,
       ceil(row_number()over(order by empno)/5.0) grp,
       empno,
       ename
from emp

```

RN	DIVISION	GRP	EMPNO	ENAME
1	.2	1	7369	SMITH
2	.4	1	7499	ALLEN
3	.6	1	7521	WARD
4	.8	1	7566	JONES
5	1	1	7654	MARTIN
6	1.2	2	7698	BLAKE
7	1.4	2	7782	CLARK
8	1.6	2	7788	SCOTT
9	1.8	2	7839	KING
10	2	2	7844	TURNER
11	2.2	3	7876	ADAMS
12	2.4	3	7900	JAMES

13	2.6	3	7902	FORD
14	2.8	3	7934	MILLER

## PostgreSQL 和 MySQL

第一步，使用标量子查询，按 EMPNO 给每行分等级：

```
select (select count(*) from emp d
       where e.empno < d.empno)+1 as rnk,
       e.empno, e.ename
from emp e
order by 1
```

RNK	EMPNO	ENAME
1	7934	MILLER
2	7902	FORD
3	7900	JAMES
4	7876	ADAMS
5	7844	TURNER
6	7839	KING
7	7788	SCOTT
8	7782	CLARK
9	7698	BLAKE
10	7654	MARTIN
11	7566	JONES
12	7521	WARD
13	7499	ALLEN
14	7369	SMITH

下一步，在 RNK 除以 5 之后，使用函数 CEIL（或 CEILING）。从逻辑上讲，除以 5 就是把行编成 5 个一组的桶，即小于等于 1 的 5 个值、大于 1 但小于等于 2 的 5 个值，另一组（包括最后 4 行，这是由于表 EMP 中的行数是 14，不是 5 的倍数）包含大于 2 但小于等于 3 的值。下面给出了做除法和使用 CEIL 的结果。可以从左到右的顺序看，从 RNK 一直到 GRP：

```
select rnk,
       rnk/5.0 as division,
       ceil(rnk/5.0) as grp,
       empno, ename
from (
select e.empno, e.ename,
       (select count(*) from emp d
        where e.empno < d.empno)+1 as rnk
from emp e
) x
order by 1
```

RNK	DIVISION	GRP	EMPNO	ENAME
1	.2	1	7934	MILLER
2	.4	1	7902	FORD
3	.6	1	7900	JAMES
4	.8	1	7876	ADAMS
5	1	1	7844	TURNER
6	1.2	2	7839	KING
7	1.4	2	7788	SCOTT
8	1.6	2	7782	CLARK
9	1.8	2	7698	BLAKE
10	2	2	7654	MARTIN
11	2.2	3	7566	JONES
12	2.4	3	7521	WARD
13	2.6	3	7499	ALLEN

14                      2.8      3    7369 SMITH

## 12.8 创建预定数目的桶

### 问题

把数据编成固定数目的桶。例如，把表 EMP 中的员工编组为 4 桶。其结果集应该如下所示：

GRP	EMPNO	ENAME
1	7369	SMITH
1	7499	ALLEN
1	7521	WARD
1	7566	JONES
2	7654	MARTIN
2	7698	BLAKE
2	7782	CLARK
2	7788	SCOTT
3	7839	KING
3	7844	TURNER
3	7876	ADAMS
4	7900	JAMES
4	7902	FORD
4	7934	MILLER

这个问题与上一节中的问题相反，对于那个问题，已知每桶的元素数，不知道桶数。而在本节中，不知道每桶的元素数，但给定了需要创建的桶的数目。

### 解决方案

如果 RDBMS 提供了创建“桶”的函数，解决方案就非常简单了。如果没有提供这样的函数，只需给每行分等级，然后在表达式中使用等级对 n 的模（n 是要创建的桶数），以确定该行落入哪个桶内。只要有可能，本节的该解决方案尽量使用 NTILE 窗口函数创建固定数目的桶。NTILE 把有序集分到指定数目的桶数中，把剩余的元素从第一桶开始再分下去。本节的结果集说明了：桶 1 和桶 2 包含 4 行，而桶 3 和桶 4 包含 3 行。如果 RDBMS 不支持 NTILE，就不用考虑哪些行在哪个桶内。本节的主要目标是创建固定数目的桶。

#### DB2

使用窗口函数 ROW\_NUMBER OVER，按 EMPNO 给行分等级，然后，使用等级跟 4 的模，创建 4 个桶：

```

1  select mod(row_number()over(order by empno),4)+1 grp,
2         empno,
3         ename
4  from emp
5  order by 1
```

#### Oracle 和 SQL Server

对于这两种数据库，DB2 解决方案也管用，不过还可以使用 NTILE 窗口函数创建 4 个桶（更方便）：

```

1 select ntile(4)over(order by empno) grp,
2        empno,
3        ename
4    from emp

```

## MySQL 和 PostgreSQL

使用自联接，按 EMPNO 给行分等级，然后，使用等级对 4 的模创建桶：

```

1 select mod(count(*),4)+1 as grp,
2        e.empno,
3        e.ename
4    from emp e, emp d
5   where e.empno >= d.empno
6   group by e.empno,e.ename
7   order by 1

```

## 讨论

### DB2

首先，使用窗口函数 ROW\_NUMBER OVER，按 EMPNO 给每行分等级：

```

select row_number()over(order by empno) grp,
       empno,
       ename
  from emp

```

GRP	EMPNO	ENAME
1	7369	SMITH
2	7499	ALLEN
3	7521	WARD
4	7566	JONES
5	7654	MARTIN
6	7698	BLAKE
7	7782	CLARK
8	7788	SCOTT
9	7839	KING
10	7844	TURNER
11	7876	ADAMS
12	7900	JAMES
13	7902	FORD
14	7934	MILLER

现在，已经把行分了等级，可使用求模函数 MOD 创建 4 个桶：

```

select mod(row_number()over(order by empno),4) grp,
       empno,
       ename
  from emp

```

GRP	EMPNO	ENAME
1	7369	SMITH
2	7499	ALLEN
3	7521	WARD
0	7566	JONES
1	7654	MARTIN
2	7698	BLAKE
3	7782	CLARK
0	7788	SCOTT
1	7839	KING
2	7844	TURNER
3	7876	ADAMS

```

0 7900 JAMES
1 7902 FORD
2 7934 MILLER

```

最后一步，给 GRP 加 1，这样桶序号就从 1 开始，而不是 0，再针对 GRP 使用 ORDER BY，按桶序号给行排序。

## Oracle 和 SQL Server

NTILE 函数会完成所有功能。只需把桶数传递给它，就能在眼前展现美妙的画面。

## MySQL 和 PostgreSQL

第一步，用表 EMP 生产笛卡儿积，使每个 EMPNO 都能够与其他 EMPNO 做比较 [下面仅列出了笛卡儿积中的一小段，整个返回结果包含 196 行(14\*14)]：

```

select e.empno,
       e.ename,
       d.empno,
       d.ename
from emp e, emp d

```

EMPNO	ENAME	EMPNO	ENAME
7369	SMITH	7369	SMITH
7369	SMITH	7499	ALLEN
7369	SMITH	7521	WARD
7369	SMITH	7566	JONES
7369	SMITH	7654	MARTIN
7369	SMITH	7698	BLAKE
7369	SMITH	7782	CLARK
7369	SMITH	7788	SCOTT
7369	SMITH	7839	KING
7369	SMITH	7844	TURNER
7369	SMITH	7876	ADAMS
7369	SMITH	7900	JAMES
7369	SMITH	7902	FORD
7369	SMITH	7934	MILLER
...			

正如这个结果集所显示的，SMITH 的 EMPNO 可以与 EMP 中所有其他员工的 EMPNO 相比较（任何员工的 EMPNO 都可以与其他所有员工的 EMPNO 相比较）。然后，仅保留笛卡儿积中 EMPNO 大于等于另一个 EMPNO 的那些行。下面列出了结果集的一部分（整个结果集包含 105 行）：

```

select e.empno,
       e.ename,
       d.empno,
       d.ename
from emp e, emp d
where e.empno >= d.empno

```

EMPNO	ENAME	EMPNO	ENAME
7934	MILLER	7934	MILLER
7934	MILLER	7902	FORD
7934	MILLER	7900	JAMES
7934	MILLER	7876	ADAMS
7934	MILLER	7844	TURNER
7934	MILLER	7839	KING

7934 MILLER	7788 SCOTT
7934 MILLER	7782 CLARK
7934 MILLER	7698 BLAKE
7934 MILLER	7654 MARTIN
7934 MILLER	7566 JONES
7934 MILLER	7521 WARD
7934 MILLER	7499 ALLEN
7934 MILLER	7369 SMITH
...	
7499 ALLEN	7499 ALLEN
7499 ALLEN	7369 SMITH
7369 SMITH	7369 SMITH

上面的输出结果中只给出了整个结果集中有关 MILLER、ALLEN 和 SMITH 的行（来自 EMP E），目的是展示如何用 WHERE 子句限定笛卡儿积的结果。由于在 WHERE 子句中对 EMPNO 的筛选使用了“大于等于”操作符，所以对于每个员工，都至少得到 1 行（每个 EMPNO 都等于它自己）。但为什么 SMITH（结果集的左侧）仅包含一行、ALLEN 包含两行，而 MILLER 包含 14 行呢？其原因是在 WHERE 子句中对 EMPNO 进行了复合计算：即“大于等于”。在 SMITH 例中，不存在比 7369 小的 EMPNO，因此只返回了 1 行；在 ALLEN 例中，很明显，ALLEN 的 EMPNO 等于它自己（所以会返回该行），而且 7499 也大于 7369（SMITH 的 EMPNO），这样 ALLEN 就会返回两行。由于 MILLER 的 EMPNO 是 7934，它比表 EMP 中所有其他的 EMPNO 都大（等于它自己），因此会返回 14 行。

现在，可以比较每个 EMPNO，并确定哪些 EMPNO 比其他 EMPNO 大。使用聚集函数 COUNT 做自联接，使结果集表现力更好：

```
select count(*) as grp,
       e.empno,
       e.ename
  from emp e, emp d
 where e.empno >= d.empno
 group by e.empno,e.ename
 order by 1
```

GRP	EMPNO	ENAME
1	7369	SMITH
2	7499	ALLEN
3	7521	WARD
4	7566	JONES
5	7654	MARTIN
6	7698	BLAKE
7	7782	CLARK
8	7788	SCOTT
9	7839	KING
10	7844	TURNER
11	7876	ADAMS
12	7900	JAMES
13	7902	FORD
14	7934	MILLER

现在，已经给行分了等级，将 GRP 对 4 的模加 1（加 1 就会使桶序号从 1 开始，而不是 0），就可创建 4 个桶，再针对 GRP 使用 ORDER BY 子句，给桶排序：

```
select mod(count(*),4)+1 as grp,
       e.empno,
       e.ename
```

```

from emp e, emp d
where e.empno >= d.empno
group by e.empno,e.ename
order by 1

```

GRP	EMPNO	ENAME
1	7900	JAMES
1	7566	JONES
1	7788	SCOTT
2	7369	SMITH
2	7902	FORD
2	7654	MARTIN
2	7839	KING
3	7499	ALLEN
3	7698	BLAKE
3	7934	MILLER
3	7844	TURNER
4	7521	WARD
4	7782	CLARK
4	7876	ADAMS

## 12.9 创建横向直方图

### 问题

使用 SQL 生成横向延伸的直方图。例如，采用横向直方图显示每个部门的职员数，一个星号 “\*” 表示一个员工。返回的结果集应该如：

DEPTNO	CNT
10	***
20	*****
30	*****

### 解决方案

这个解决方案的关键是使用聚集函数 COUNT，并使用 GROUP BY DEPTNO 确定每个 DEPTNO 中的员工数。然后把 COUNT 的返回值传递给字符串函数，该函数将生成一系列 “\*” 字符。

### DB2

用 REPEAT 函数生成直方图：

```

1 select deptno,
2        repeat('*',count(*)) cnt
3   from emp
4  group by deptno

```

### Oracle、PostgreSQL 和 MySQL

用 LPAD 函数生成所需的 “\*” 串：

```

1 select deptno,
2        lpad('*',count(*),'*') as cnt
3   from emp
4  group by deptno

```



## SQL Server

用 REPLICATE 函数生成直方图：

```
1 select deptno,  
2        replicate(' ',count(*)) cnt  
3   from emp  
4  group by deptno
```

## 讨论

对于所有数据库，解决问题的技巧都相同。唯一的差别是用来产生代表员工的“\*”字符串的函数不同。下面的讨论将采用 Oracle 解决方案，但其说明适用于所有解决方案。

第一步，计算每个部门的员工数：

```
select deptno,  
       count(*)  
  from emp  
 group by deptno
```

DEPTNO	COUNT(*)
10	3
20	5
30	6

然后，使用 COUNT(\*) 返回的值设置每个部门包含的“\*”字符数。把 COUNT(\*) 值作为参数传递给字符串函数 LPAD，以返回相应数目的“\*”：

```
select deptno,  
       lpad(' ',count(*),' ') as cnt  
  from emp  
 group by deptno
```

DEPTNO	CNT
10	***
20	*****
30	*****

对于 PostgreSQL 用户，需要把 COUNT(\*) 转换为整型，如下所示：

```
select deptno,  
       lpad(' ',count(*):: integer,' ') as cnt  
  from emp  
 group by deptno
```

DEPTNO	CNT
10	***
20	*****
30	*****

这个 CAST 是必须要有的，因为 PostgreSQL 要求 LPAD 参数为整型。



13 order by 1 desc, 2 desc, 3 desc

讨论

DB2、Oracle 和 SQL Server

使用窗口函数ROW\_NUMBER, 唯一表示每个DEPTNO中的每个“\*”实例。使用CASE表达式, 对于每个部门的每个员工都返回一个“\*”:

```
select row_number()over(partition by deptno order by empno) rn,
       case when deptno=10 then '*' else null end deptno_10,
       case when deptno=20 then '*' else null end deptno_20,
       case when deptno=30 then '*' else null end deptno_30
from emp
```

RN	DEPTNO_10	DEPTNO_20	DEPTNO_30
1	*		
2	*		
3	*		
1		*	
2		*	
3		*	
4		*	
5		*	
1			*
2			*
3			*
4			*
5			*
6			*

下一步, 也就是最后一步, 是针对每个CASE表达式使用聚集函数MAX, 按RN分组, 以便去除结果集中的NULL值。根据RDBMS对NULL排序的方式决定按升序(ASC)或降序(DESC)给结果集排序:

```
select max(deptno_10) d10,
       max(deptno_20) d20,
       max(deptno_30) d30
from (
select row_number()over(partition by deptno order by empno) rn,
       case when deptno=10 then '*' else null end deptno_10,
       case when deptno=20 then '*' else null end deptno_20,
       case when deptno=30 then '*' else null end deptno_30
from emp
) x
group by rn
order by 1 desc, 2 desc, 3 desc
```

D10	D20	D30
		*
	*	*
	*	*
*	*	*
*	*	*
*	*	*

PostgreSQL 和 MySQL

首先, 使用标量子查询, 唯一表示每个部门中的每个“\*”实例。对于每个DEPTNO, 该

标量子查询按 EMPNO 给员工分等级，这样就不会存在重复。用 CASE 表达式为每个部门的每个员工生成一个 “\*”：

```
select case when e.deptno=10 then '*' else null end deptno_10,
       case when e.deptno=20 then '*' else null end deptno_20,
       case when e.deptno=30 then '*' else null end deptno_30,
       (select count(*) from emp d
        where e.deptno=d.deptno and e.empno < d.empno ) as rnk
from emp e
```

DEPTNO_10	DEPTNO_20	DEPTNO_30	RNK
	*		4
		*	5
		*	4
	*		3
		*	3
		*	2
*			2
	*		2
*			1
		*	1
	*		1
		*	0
	*		0
*			0

然后，针对每个 CASE 表达式使用聚集函数 MAX，以去除结果集中的 NULL 值，确保按 RNK 分组。根据 RDBMS 对 NULL 排序的方式决定按升序 (ASC) 或降序 (DESC) 给结果集排序：

```
select max(deptno_10) as d10,
       max(deptno_20) as d20,
       max(deptno_30) as d30
from (
select case when e.deptno=10 then '*' else null end deptno_10,
       case when e.deptno=20 then '*' else null end deptno_20,
       case when e.deptno=30 then '*' else null end deptno_30,
       (select count(*) from emp d
        where e.deptno=d.deptno and e.empno < d.empno ) as rnk
from emp e
) x
group by rnk
order by 1 desc, 2 desc, 3 desc
```

D10	D20	D30
		*
	*	*
	*	*
*	*	*
*	*	*
*	*	*

## 12.11 返回未包含在 GROUP BY 中的列问题

正在执行一个 GROUP BY 查询，并希望返回那些属于选择列表而不包含于 GROUP BY 子句中的列。通常，这是不可能的，因为对于这样的非组列，并不是每行都包含唯一值。

假设要找到每个部门中工资最高和最低的员工,以及每种职位中工资最高和最低的员工,要查看这些人的姓名、所在部门、职位名称以及工资。希望返回的结果集如下:

DEPTNO	ENAME	JOB	SAL	DEPT_STATUS	JOB_STATUS
10	MILLER	CLERK	1300	LOW SAL IN DEPT	TOP SAL IN JOB
10	CLARK	MANAGER	2450		LOW SAL IN JOB
10	KING	PRESIDENT	5000	TOP SAL IN DEPT	TOP SAL IN JOB
20	SCOTT	ANALYST	3000	TOP SAL IN DEPT	TOP SAL IN JOB
20	FORD	ANALYST	3000	TOP SAL IN DEPT	TOP SAL IN JOB
20	SMITH	CLERK	800	LOW SAL IN DEPT	LOW SAL IN JOB
20	JONES	MANAGER	2975		TOP SAL IN JOB
30	JAMES	CLERK	950	LOW SAL IN DEPT	
30	MARTIN	SALESMAN	1250		LOW SAL IN JOB
30	WARD	SALESMAN	1250		LOW SAL IN JOB
30	ALLEN	SALESMAN	1600		TOP SAL IN JOB
30	BLAKE	MANAGER	2850	TOP SAL IN DEPT	

令人遗憾的是, SELECT 子句中的所有列都放到 GROUP BY 字句中会破坏分组。请看下面的例子。员工“KING”工资最高,采用下列查询进行验证:

```
select ename,max(sal)
  from emp
 group by ename
```

上面的查询不会返回“KING”及 KING 的工资,而是返回表 EMP 中包含的所有 14 行记录。这是分组引起的: MAX(SAL) 分别作用于每个 ENAME。因此,上面的查询似乎是要“找出工资最高的员工”,而事实上它所做的是“找出表 EMP 中每个 ENAME 的最高工资”。本节说明了列出 ENAME 而不必按该列分组 (GROUP BY) 的技巧。

## 解决方案

使用内联视图,按 DEPTNO 和 JOB 找到最高工资和最低工资。然后,只保留工资最高或工资最低的员工。

### DB2、Oracle 和 SQL Server

使用窗口函数 MAX OVER 和 MIN OVER,按 DEPTNO 和 JOB 找到高工资和低工资。然后,只保留最高工资或最低工资的行:

```
1  select deptno,ename,job,sal,
2         case when sal = max_by_dept
3              then 'TOP SAL IN DEPT'
4              when sal = min_by_dept
5              then 'LOW SAL IN DEPT'
6         end dept_status,
7         case when sal = max_by_job
8              then 'TOP SAL IN JOB'
9              when sal = min_by_job
10             then 'LOW SAL IN JOB'
11        end job_status
12  from (
13  select deptno,ename,job,sal,
14         max(sal)over(partition by deptno) max_by_dept,
15         max(sal)over(partition by job)    max_by_job,
16         min(sal)over(partition by deptno) min_by_dept,
```

```

17      min(sal)over(partition by job)      min_by_job
18  from emp
19      ) emp_sals
20  where sal in (max_by_dept,max_by_job,
21               min_by_dept,min_by_job)

```

## PostgreSQL 和 MySQL

使用标量子查询，按 DEPTNO 和 JOB 找到最高工资和最低工资。然后，只保留与这些工资相匹配的员工：

```

1  select deptno,ename,job,sal,
2         case when sal = max_by_dept
3              then 'TOP SAL IN DEPT'
4              when sal = min_by_dept
5              then 'LOW SAL IN DEPT'
6         end as dept_status,
7         case when sal = max_by_job
8              then 'TOP SAL IN JOB'
9              when sal = min_by_job
10             then 'LOW SAL IN JOB'
11        end as job_status
12  from (
13  select e.deptno,e.ename,e.job,e.sal,
14         (select max(sal) from emp d
15          where d.deptno = e.deptno) as max_by_dept,
16         (select max(sal) from emp d
17          where d.job = e.job) as max_by_job,
18         (select min(sal) from emp d
19          where d.deptno = e.deptno) as min_by_dept,
20         (select min(sal) from emp d
21          where d.job = e.job) as min_by_job
22  from emp e
23      ) x
24  where sal in (max_by_dept,max_by_job,
25               min_by_dept,min_by_job)

```

## 讨论

### DB2、Oracle 和 SQL Server

第一步，使用窗口函数 MAX OVER 和 MIN OVER，按 DEPTNO 和 JOB 找到最高工资和最低工资：

```

select deptno,ename,job,sal,
       max(sal)over(partition by deptno) maxDEPT,
       max(sal)over(partition by job)    maxJOB,
       min(sal)over(partition by deptno) minDEPT,
       min(sal)over(partition by job)    minJOB
from emp

```

DEPTNO	ENAME	JOB	SAL	MAXDEPT	MAXJOB	MINDEPT	MINJOB
10	MILLER	CLERK	1300	5000	1300	1300	800
10	CLARK	MANAGER	2450	5000	2975	1300	2450
10	KING	PRESIDENT	5000	5000	5000	1300	5000
20	SCOTT	ANALYST	3000	3000	3000	800	3000
20	FORD	ANALYST	3000	3000	3000	800	3000
20	SMITH	CLERK	800	3000	1300	800	800
20	JONES	MANAGER	2975	3000	2975	800	2450
20	ADAMS	CLERK	1100	3000	1300	800	800
30	JAMES	CLERK	950	2850	1300	950	800

30	MARTIN	SALESMAN	1250	2850	1600	950	1250
30	TURNER	SALESMAN	1500	2850	1600	950	1250
30	WARD	SALESMAN	1250	2850	1600	950	1250
30	ALLEN	SALESMAN	1600	2850	1600	950	1250
30	BLAKE	MANAGER	2850	2850	2975	950	2450

至此，就可以把每个员工工资跟相应DEPTNO和JOB的最高工资和最低工资做比较。应该注意，分组（在SELECT子句中包含多列）不会影响MIN OVER 和 MAX OVER 返回的值。这就是窗口函数的优点：根据定义的“组”或分区计算聚集，并为每一组返回多行。最后一步，只需把窗口函数“包”成内联视图，并且仅保留那些与窗口函数返回的值相匹配的行。使用简单的CASE表达式，显示最终结果集中每个员工的“状态”：

```
select deptno,ename,job,sal,
       case when sal = max_by_dept
            then 'TOP SAL IN DEPT'
            when sal = min_by_dept
            then 'LOW SAL IN DEPT'
            end dept_status,
       case when sal = max_by_job
            then 'TOP SAL IN JOB'
            when sal = min_by_job
            then 'LOW SAL IN JOB'
            end job_status
from (
select deptno,ename,job,sal,
       max(sal)over(partition by deptno) max_by_dept,
       max(sal)over(partition by job) max_by_job,
       min(sal)over(partition by deptno) min_by_dept,
       min(sal)over(partition by job) min_by_job
from emp
) x
where sal in (max_by_dept,max_by_job,
             min_by_dept,min_by_job)
```

DEPTNO	ENAME	JOB	SAL	DEPT_STATUS	JOB_STATUS
10	MILLER	CLERK	1300	LOW SAL IN DEPT	TOP SAL IN JOB
10	CLARK	MANAGER	2450	LOW SAL IN JOB	TOP SAL IN DEPT
10	KING	PRESIDENT	5000	TOP SAL IN DEPT	TOP SAL IN JOB
20	SCOTT	ANALYST	3000	TOP SAL IN DEPT	TOP SAL IN JOB
20	FORD	ANALYST	3000	TOP SAL IN DEPT	TOP SAL IN JOB
20	SMITH	CLERK	800	LOW SAL IN DEPT	LOW SAL IN JOB
20	JONES	MANAGER	2975	TOP SAL IN JOB	TOP SAL IN DEPT
30	JAMES	CLERK	950	LOW SAL IN DEPT	TOP SAL IN JOB
30	MARTIN	SALESMAN	1250	LOW SAL IN JOB	TOP SAL IN DEPT
30	WARD	SALESMAN	1250	LOW SAL IN JOB	TOP SAL IN DEPT
30	ALLEN	SALESMAN	1600	TOP SAL IN DEPT	TOP SAL IN JOB
30	BLAKE	MANAGER	2850	TOP SAL IN DEPT	TOP SAL IN JOB

## PostgreSQL 和 MySQL

第一步，使用标量子查询，按DEPTNO和JOB找到最高工资和最低工资：

```
select e.deptno,e.ename,e.job,e.sal,
       (select max(sal) from emp d
        where d.deptno = e.deptno) as maxDEPT,
       (select max(sal) from emp d
        where d.job = e.job) as maxJOB,
       (select min(sal) from emp d
        where d.deptno = e.deptno) as minDEPT,
       (select min(sal) from emp d
        where d.job = e.job) as minJOB
```

```

from emp e
DEPTNO ENAME JOB SAL MAXDEPT MAXJOB MINDEPT MINJOB
-----
20 SMITH CLERK 800 3000 1300 800 800
30 ALLEN SALESMAN 1600 2850 1600 950 1250
30 WARD SALESMAN 1250 2850 1600 950 1250
20 JONES MANAGER 2975 3000 2975 800 2450
30 MARTIN SALESMAN 1250 2850 1600 950 1250
30 BLAKE MANAGER 2850 2850 2975 950 2450
10 CLARK MANAGER 2450 5000 2975 1300 2450
20 SCOTT ANALYST 3000 3000 3000 800 3000
10 KING PRESIDENT 5000 5000 5000 1300 5000
30 TURNER SALESMAN 1500 2850 1600 950 1250
20 ADAMS CLERK 1100 3000 1300 800 800
30 JAMES CLERK 950 2850 1300 950 800
20 FORD ANALYST 3000 3000 3000 800 3000
10 MILLER CLERK 1300 5000 1300 1300 800

```

现在可以将各 DEPTNO 和 JOB 的最高工资和最低工资与表 EMP 中的其他所有工资相比了。最后一步，只要把该标量子查询“包”成内联视图中，并且仅保留那些工资与标量子查询返回的值相匹配的员工。用 ASE 表达式显示最终结果集中每个员工的状态：

```

select deptno,ename,job,sal,
       case when sal = max_by_dept
         then 'TOP SAL IN DEPT'
         when sal = min_by_dept
         then 'LOW SAL IN DEPT'
       end as dept_status,
       case when sal = max_by_job
         then 'TOP SAL IN JOB'
         when sal = min_by_job
         then 'LOW SAL IN JOB'
       end as job_status
from (
select e.deptno,e.ename,e.job,e.sal,
       (select max(sal) from emp d
        where d.deptno = e.deptno) as max_by_dept,
       (select max(sal) from emp d
        where d.job = e.job) as max_by_job,
       (select min(sal) from emp d
        where d.deptno = e.deptno) as min_by_dept,
       (select min(sal) from emp d
        where d.job = e.job) as min_by_job
from emp e
) x
where sal in (max_by_dept,max_by_job,
             min_by_dept,min_by_job)
DEPTNO ENAME JOB SAL DEPT_STATUS JOB_STATUS
-----
10 CLARK MANAGER 2450 LOW SAL IN JOB
10 KING PRESIDENT 5000 TOP SAL IN DEPT TOP SAL IN JOB
10 MILLER CLERK 1300 LOW SAL IN DEPT TOP SAL IN JOB
20 SMITH CLERK 800 LOW SAL IN DEPT LOW SAL IN JOB
20 FORD ANALYST 3000 TOP SAL IN DEPT TOP SAL IN JOB
20 SCOTT ANALYST 3000 TOP SAL IN DEPT TOP SAL IN JOB
20 JONES MANAGER 2975 TOP SAL IN JOB
30 ALLEN SALESMAN 1600 TOP SAL IN JOB
30 BLAKE MANAGER 2850 TOP SAL IN DEPT
30 MARTIN SALESMAN 1250 LOW SAL IN JOB
30 JAMES CLERK 950 LOW SAL IN DEPT
30 WARD SALESMAN 1250 LOW SAL IN JOB

```



## 12.12 计算简单的小计

### 问题

本节的目的：定义一个“简单小计”结果集，它包含一系列的聚集值以及全表的总计值。例如将表 EMP 中各 JOB 的工资总和放入一个结果集，并且将表 EMP 中所有工资的总和也加入其中。按 JOB 的工资和是小计，而表 EMP 中的工资总和是总计。这样的结果集应该如下：

JOB	SAL
ANALYST	6000
CLERK	4150
MANAGER	8275
PRESIDENT	5000
SALESMAN	5600
TOTAL	29025

### 解决方案

GROUP BY 子句的 ROLLUP 扩展可以完美地解决这个问题。如果 RDBMS 没有提供 ROLLUP，则可以使用标量子查询或 UNION 查询来解决这个问题，但可能会比这难一些。

#### DB2 和 Oracle

使用聚集函数 SUM 计算工资的和，并使用 GROUP BY 的 ROLLUP 扩展把结果分组为小计（按 JOB）及总计（对整个表）：

```
1 select case grouping(job)
2           when 0 then job
3           else 'TOTAL'
4         end job,
5         sum(sal) sal
6   from emp
7  group by rollup(job)
```

#### SQL Server 和 MySQL

使用聚集函数 SUM 计算工资总和，再使用 WITH ROLLUP，把结果分组为小计（按 JOB）和总计（对整个表而言）。然后，使用 COALESCE 给总计行加标签 ‘TOTAL’（否则，JOB 列的值就为 NULL）：

```
1 select coalesce(job, 'TOTAL') job,
2        sum(sal) sal
3   from emp
4  group by job with rollup
```

对 SQL Server 还有另一种方法，即不使用 COALESCE 函数，而使用 Oracle/DB2 方案中介绍的 GROUPING 函数来确定聚集的层次。

## PostgreSQL

使用聚集函数 SUM，按 DEPTNO 计算工资和。然后，在查询中使用 UNION ALL，生成表中所有工资的总和：

```
1 select job, sum(sal) as sal
2   from emp
3   group by job
4   union all
5  select 'TOTAL', sum(sal)
6   from emp
```

## 讨论

### DB2 和 Oracle

首先，使用聚集函数 SUM，并按 JOB 分组，以便按 JOB 计算工资和：

```
select job, sum(sal) sal
  from emp
 group by job
```

JOB	SAL
ANALYST	6000
CLERK	4150
MANAGER	8275
PRESIDENT	5000
SALESMAN	5600

下一步，使用 GROUP BY 的 ROLLUP 扩展，生成所有工资的总计以及每个 JOB 的小计：

```
select job, sum(sal) sal
  from emp
 group by rollup(job)
```

JOB	SAL
ANALYST	6000
CLERK	4150
MANAGER	8275
PRESIDENT	5000
SALESMAN	5600
	29025

最后一步，对 JOB 列使用 GROUPING 函数，以便显示总计的标签。如果 JOB 值为 NULL，则 GROUPING 函数将返回 1，这标志着 SAL 值是由 ROLLUP 创建的总计；如果 JOB 值不为 NULL，则 GROUPING 函数将返回 0，这标志着 SAL 值是由 GROUP BY 创建的，而不是 ROLLUP 创建的。在 CASE 表达式中加入 GROUPING(JOB) 的调用，就会返回正确的职位名称或标签 'TOTAL'：

```
select case grouping(job)
       when 0 then job
       else 'TOTAL'
       end job,
       sum(sal) sal
  from emp
 group by rollup(job)
```

JOB	SAL
-----	-----
ANALYST	6000
CLERK	4150
MANAGER	8275
PRESIDENT	5000
SALESMAN	5600
TOTAL	29025

## SQL Server 和 MySQL

首先，使用聚集函数 SUM，并按 JOB 分组，以便按 JOB 计算工资和：

```
select job, sum(sal) sal
  from emp
 group by job
```

JOB	SAL
-----	-----
ANALYST	6000
CLERK	4150
MANAGER	8275
PRESIDENT	5000
SALESMAN	5600

下一步，使用 GROUP BY 的 ROLLUP 扩展，生成所有工资的总计以及每个 JOB 的小计：

```
select job, sum(sal) sal
  from emp
 group by job with rollup
```

JOB	SAL
-----	-----
ANALYST	6000
CLERK	4150
MANAGER	8275
PRESIDENT	5000
SALESMAN	5600
	29025

最后一步，对 JOB 列使用 COALESCE 函数。如果 JOB 值为 NULL，则 SAL 值就是由 ROLLUP 创建的总计；如果 JOB 值不为 NULL，则 SAL 值就是由“常规的” GROUP BY 得到的结果，而不是 ROLLUP 的结果：

```
select coalesce(job, 'TOTAL') job,
       sum(sal) sal
  from emp
 group by job with rollup
```

JOB	SAL
-----	-----
ANALYST	6000
CLERK	4150
MANAGER	8275
PRESIDENT	5000
SALESMAN	5600
TOTAL	29025

## PostgreSQL

第一步，按 JOB 给结果分组，使用聚集函数 SUM 返回按 JOB 计算的工资和：

```
select job, sum(sal) sal
```

```

      from emp
      group by job
JOB          SAL
-----
ANALYST      6000
CLERK        4150
MANAGER      8275
PRESIDENT    5000
SALESMAN     5600

```

最后一步，使用 UNION ALL，计算上述查询的总计：

```

select job, sum(sal) as sal
  from emp
  group by job
 union all
select 'TOTAL', sum(sal)
  from emp
JOB          SAL
-----
ANALYST      6000
CLERK        4150
MANAGER      8275
PRESIDENT    5000
SALESMAN     5600
TOTAL        29025

```

## 12.13 计算所有表达式组合的小计

### 问题

对 JOB/DEPTNO 的每种组合，求按 DEPTNO 和 JOB 的总工资。并求表 EMP 中所有工资的总计。返回的结果集应如：

DEPTNO	JOB	CATEGORY	SAL
10	CLERK	TOTAL BY DEPT AND JOB	1300
10	MANAGER	TOTAL BY DEPT AND JOB	2450
10	PRESIDENT	TOTAL BY DEPT AND JOB	5000
20	CLERK	TOTAL BY DEPT AND JOB	1900
30	CLERK	TOTAL BY DEPT AND JOB	950
30	SALESMAN	TOTAL BY DEPT AND JOB	5600
30	MANAGER	TOTAL BY DEPT AND JOB	2850
20	MANAGER	TOTAL BY DEPT AND JOB	2975
20	ANALYST	TOTAL BY DEPT AND JOB	6000
	CLERK	TOTAL BY JOB	4150
	ANALYST	TOTAL BY JOB	6000
	MANAGER	TOTAL BY JOB	8275
	PRESIDENT	TOTAL BY JOB	5000
	SALESMAN	TOTAL BY JOB	5600
10		TOTAL BY DEPT	8750
30		TOTAL BY DEPT	9400
20		TOTAL BY DEPT	10875
		GRAND TOTAL FOR TABLE	29025

### 解决方案

最近几年，GROUP BY 中增加的扩展使这个问题相当容易解决。如果使用的平台没有提供这种计算各层小计的扩展，那么必须用自联接或标量子查询计算。

## DB2

对于 DB2，必须使用 CAST，把结果从 GROUPING 格式转换为 CHAR(1)数据类型：

```
1 select deptno,
2        job,
3        case cast(grouping(deptno) as char(1))||
4              cast(grouping(job) as char(1))
5              when '00' then 'TOTAL BY DEPT AND JOB'
6              when '10' then 'TOTAL BY JOB'
7              when '01' then 'TOTAL BY DEPT'
8              when '11' then 'TOTAL FOR TABLE'
9        end category,
10       sum(sal)
11 from emp
12 group by cube(deptno,job)
13 order by grouping(job),grouping(deptno)
```

## Oracle

使用 GROUP BY 子句中的 CUBE 扩展及连接操作符 “||”：

```
1 select deptno,
2        job,
3        case grouping(deptno)||grouping(job)
4              when '00' then 'TOTAL BY DEPT AND JOB'
5              when '10' then 'TOTAL BY JOB'
6              when '01' then 'TOTAL BY DEPT'
7              when '11' then 'GRAND TOTAL FOR TABLE'
8        end category,
9        sum(sal) sal
10 from emp
11 group by cube(deptno,job)
12 order by grouping(job),grouping(deptno)
```

## SQL Server

使用 GROUP BY 子句的 CUBE 扩展。对于 SQL Server，必须用 CAST 把 GROUPING 的结果转换为 CHAR(1)类型，而且需要使用 “+” 操作符进行连接（与 Oracle 的 “||” 操作符不同）：

```
1 select deptno,
2        job,
3        case cast(grouping(deptno)as char(1))+
4              cast(grouping(job)as char(1))
5              when '00' then 'TOTAL BY DEPT AND JOB'
6              when '10' then 'TOTAL BY JOB'
7              when '01' then 'TOTAL BY DEPT'
8              when '11' then 'GRAND TOTAL FOR TABLE'
9        end category,
10       sum(sal) sal
11 from emp
12 group by deptno,job with cube
13 order by grouping(job),grouping(deptno)
```

## PostgreSQL 和 MySQL

使用多个 UNION ALL，各自产生不同的和：

```
1 select deptno, job,
2        'TOTAL BY DEPT AND JOB' as category,
3        sum(sal) as sal
```

```

4      from emp
5      group by deptno, job
6      union all
7      select null, job, 'TOTAL BY JOB', sum(sal)
8      from emp
9      group by job
10     union all
11     select deptno, null, 'TOTAL BY DEPT', sum(sal)
12     from emp
13     group by deptno
14     union all
15     select null,null,'GRAND TOTAL FOR TABLE', sum(sal)
16     from emp

```

## 讨论

### Oracle、DB2 和 SQL Server

从本质上讲，这三个解决方案相同。首先，使用聚集函数 SUM，并按 DEPTNO 和 JOB 分组，以求得每个 JOB 和 DEPTNO 组合的总工资：

```

select deptno, job, sum(sal) sal
  from emp
 group by deptno, job

```

DEPTNO	JOB	SAL
10	CLERK	1300
10	MANAGER	2450
10	PRESIDENT	5000
20	CLERK	1900
20	ANALYST	6000
20	MANAGER	2975
30	CLERK	950
30	MANAGER	2850
30	SALESMAN	5600

下一步，创建按 JOB 和 DEPTNO 分组的小计及整个表的总计。使用 GROUP BY 子句的 CUBE 扩展，以便完成按 DEPTNO 和 JOB 计算 SAL 的和，并对总表求和：

```

select deptno,
       job,
       sum(sal) sal
  from emp
 group by cube(deptno,job)

```

DEPTNO	JOB	SAL
		29025
	CLERK	4150
	ANALYST	6000
	MANAGER	8275
	SALESMAN	5600
	PRESIDENT	5000
10		8750
10	CLERK	1300
10	MANAGER	2450
10	PRESIDENT	5000
20		10875
20	CLERK	1900
20	ANALYST	6000
20	MANAGER	2975
30		9400

```

30 CLERK          950
30 MANAGER        2850
30 SALESMAN       5600
    
```

下一步，可使用 GROUPING 函数和 CASE，把结果设置为更有意义的输出格式。GROUPING(JOB)返回的值将为1或0，这取决于SAL值是来自 GROUP BY 还是 CUBE。如果其结果来自 CUBE，则值将为1，否则为0。GROUPING(DEPTNO)的返回值也是如此。查阅该解决方案的第一步就可以看出：分组是按DEPTNO和JOB进行的。这样，当某个行是DEPTNO和JOB的组合时，从GROUPING调用返回的值为0。下面的查询证实了这种说法：

```

select deptno,
       job,
       grouping(deptno) is_deptno_subtotal,
       grouping(job) is_job_subtotal,
       sum(sal) sal
from emp
group by cube(deptno,job)
order by 3,4
    
```

DEPTNO	JOB	IS_DEPTNO_SUBTOTAL	IS_JOB_SUBTOTAL	SAL
10	CLERK	0	0	1300
10	MANAGER	0	0	2450
10	PRESIDENT	0	0	5000
20	CLERK	0	0	1900
30	CLERK	0	0	950
30	SALESMAN	0	0	5600
30	MANAGER	0	0	2850
20	MANAGER	0	0	2975
20	ANALYST	0	0	6000
10		0	1	8750
20		0	1	10875
30		0	1	9400
	CLERK	1	0	4150
	ANALYST	1	0	6000
	MANAGER	1	0	8275
	PRESIDENT	1	0	5000
	SALESMAN	1	0	5600
		1	1	29025

最后一步，使用CASE表达式，确定每行属于哪个类别，这取决于GROUPING(JOB)和GROUPING(DEPTNO)返回的值：

```

select deptno,
       job,
       case grouping(deptno)||grouping(job)
         when '00' then 'TOTAL BY DEPT AND JOB'
         when '10' then 'TOTAL BY JOB'
         when '01' then 'TOTAL BY DEPT'
         when '11' then 'GRAND TOTAL FOR TABLE'
       end category,
       sum(sal) sal
from emp
group by cube(deptno,job)
order by grouping(job),grouping(deptno)
    
```

DEPTNO	JOB	CATEGORY	SAL
10	CLERK	TOTAL BY DEPT AND JOB	1300
10	MANAGER	TOTAL BY DEPT AND JOB	2450

10	PRESIDENT	TOTAL BY DEPT AND JOB	5000
20	CLERK	TOTAL BY DEPT AND JOB	1900
30	CLERK	TOTAL BY DEPT AND JOB	950
30	SALESMAN	TOTAL BY DEPT AND JOB	5600
30	MANAGER	TOTAL BY DEPT AND JOB	2850
20	MANAGER	TOTAL BY DEPT AND JOB	2975
20	ANALYST	TOTAL BY DEPT AND JOB	6000
	CLERK	TOTAL BY JOB	4150
	ANALYST	TOTAL BY JOB	6000
	MANAGER	TOTAL BY JOB	8275
	PRESIDENT	TOTAL BY JOB	5000
	SALESMAN	TOTAL BY JOB	5600
10		TOTAL BY DEPT	8750
30		TOTAL BY DEPT	9400
20		TOTAL BY DEPT	10875
		GRAND TOTAL FOR TABLE	29025

在 Oracle 解决方案中，隐式地把 GROUPING 函数的结果转换为字符类型，为连接两个值做准备。DB2 和 SQL Server 用户必须像相应的方案中一样，显式地进行数据类型把 GROUPING 函数返回的结果转换为 CHAR(1)。另外，SQL Server 用户一定要使用 “+” 操作符，而不是 “||” 操作符，把来自两个 GROUPING 调用的结果连接成一个字符串。

对于 Oracle 和 DB2 用户，可以使用 GROUP BY 的另一个扩展，即 GROUPING SETS。该扩展非常有用。例如，可以使用 GROUPING SETS 模拟 CUBE 创建的输出，如下所示 (DB2 和 SQL Server 用户必须对 GROUPING 函数返回的值使用 CAST，就像 CUBE 解决方案一样)：

```
select deptno,
       job,
       case grouping(deptno)||grouping(job)
         when '00' then 'TOTAL BY DEPT AND JOB'
         when '10' then 'TOTAL BY JOB'
         when '01' then 'TOTAL BY DEPT'
         when '11' then 'GRAND TOTAL FOR TABLE'
       end category,
       sum(sal) sal
from emp
group by grouping sets ((deptno),(job),(deptno,job),())
```

DEPTNO	JOB	CATEGORY	SAL
10	CLERK	TOTAL BY DEPT AND JOB	1300
20	CLERK	TOTAL BY DEPT AND JOB	1900
30	CLERK	TOTAL BY DEPT AND JOB	950
20	ANALYST	TOTAL BY DEPT AND JOB	6000
10	MANAGER	TOTAL BY DEPT AND JOB	2450
20	MANAGER	TOTAL BY DEPT AND JOB	2975
30	MANAGER	TOTAL BY DEPT AND JOB	2850
30	SALESMAN	TOTAL BY DEPT AND JOB	5600
10	PRESIDENT	TOTAL BY DEPT AND JOB	5000
	CLERK	TOTAL BY JOB	4150
	ANALYST	TOTAL BY JOB	6000
	MANAGER	TOTAL BY JOB	8275
	SALESMAN	TOTAL BY JOB	5600
	PRESIDENT	TOTAL BY JOB	5000
10		TOTAL BY DEPT	8750
20		TOTAL BY DEPT	10875
30		TOTAL BY DEPT	9400
		GRAND TOTAL FOR TABLE	29025

GROUPING SETS 的优点在于允许定义组。上述查询中的 GROUPING SETS 子句会导



致按 DEPTNO、JOB 或 DEPTNO 和 JOB 组合创建组，最后，空括号会请求总计。当需要采用不同层的聚集创建报表时，GROUPING SETS 会提供很大的灵活性。例如，如果想要修改上述例子，使它不包含 GRAND TOTAL，则简单地修改 GROUPING SETS 子句、去掉空括号即可：

```
/* no grand total */
select deptno,
       job,
       case grouping(deptno)||grouping(job)
         when '00' then 'TOTAL BY DEPT AND JOB'
         when '10' then 'TOTAL BY JOB'
         when '01' then 'TOTAL BY DEPT'
         when '11' then 'GRAND TOTAL FOR TABLE'
       end category,
       sum(sal) sal
from emp
group by grouping sets ((deptno),(job),(deptno,job))
```

DEPTNO	JOB	CATEGORY	SAL
10	CLERK	TOTAL BY DEPT AND JOB	1300
20	CLERK	TOTAL BY DEPT AND JOB	1900
30	CLERK	TOTAL BY DEPT AND JOB	950
20	ANALYST	TOTAL BY DEPT AND JOB	6000
10	MANAGER	TOTAL BY DEPT AND JOB	2450
20	MANAGER	TOTAL BY DEPT AND JOB	2975
30	MANAGER	TOTAL BY DEPT AND JOB	2850
30	SALESMAN	TOTAL BY DEPT AND JOB	5600
10	PRESIDENT	TOTAL BY DEPT AND JOB	5000
	CLERK	TOTAL BY JOB	4150
	ANALYST	TOTAL BY JOB	6000
	MANAGER	TOTAL BY JOB	8275
	SALESMAN	TOTAL BY JOB	5600
	PRESIDENT	TOTAL BY JOB	5000
10		TOTAL BY DEPT	8750
20		TOTAL BY DEPT	10875
30		TOTAL BY DEPT	9400

也可以去掉小计，例如，针对 DEPTNO 的小计，只需从 GROUPING SETS 子句中删除 (DEPTNO) 即可：

```
/* no subtotals by DEPTNO */
select deptno,
       job,
       case grouping(deptno)||grouping(job)
         when '00' then 'TOTAL BY DEPT AND JOB'
         when '10' then 'TOTAL BY JOB'
         when '01' then 'TOTAL BY DEPT'
         when '11' then 'GRAND TOTAL FOR TABLE'
       end category,
       sum(sal) sal
from emp
group by grouping sets ((job),(deptno,job),())
order by 3
```

DEPTNO	JOB	CATEGORY	SAL
		GRAND TOTAL FOR TABLE	29025
10	CLERK	TOTAL BY DEPT AND JOB	1300
20	CLERK	TOTAL BY DEPT AND JOB	1900
30	CLERK	TOTAL BY DEPT AND JOB	950

20	ANALYST	TOTAL BY DEPT AND JOB	6000
20	MANAGER	TOTAL BY DEPT AND JOB	2975
30	MANAGER	TOTAL BY DEPT AND JOB	2850
30	SALESMAN	TOTAL BY DEPT AND JOB	5600
10	PRESIDENT	TOTAL BY DEPT AND JOB	5000
10	MANAGER	TOTAL BY DEPT AND JOB	2450
	CLERK	TOTAL BY JOB	4150
	SALESMAN	TOTAL BY JOB	5600
	PRESIDENT	TOTAL BY JOB	5000
	MANAGER	TOTAL BY JOB	8275
	ANALYST	TOTAL BY JOB	6000

可以看到, GROUPING SETS 确实使计算总计和小计非常容易, 这样, 就可以从不同角度查看自己的数据。

## PostgreSQL 和 MySQL

第一步, 使用聚集函数 SUM, 并按 DEPTNO 和 JOB 分组:

```
select deptno, job,
       'TOTAL BY DEPT AND JOB' as category,
       sum(sal) as sal
from emp
group by deptno, job
```

DEPTNO	JOB	CATEGORY	SAL
10	CLERK	TOTAL BY DEPT AND JOB	1300
10	MANAGER	TOTAL BY DEPT AND JOB	2450
10	PRESIDENT	TOTAL BY DEPT AND JOB	5000
20	CLERK	TOTAL BY DEPT AND JOB	1900
20	ANALYST	TOTAL BY DEPT AND JOB	6000
20	MANAGER	TOTAL BY DEPT AND JOB	2975
30	CLERK	TOTAL BY DEPT AND JOB	950
30	MANAGER	TOTAL BY DEPT AND JOB	2850
30	SALESMAN	TOTAL BY DEPT AND JOB	5600

下一步, 使用 UNION ALL, 按 JOB 计算所有工资的总和:

```
select deptno, job,
       'TOTAL BY DEPT AND JOB' as category,
       sum(sal) as sal
from emp
group by deptno, job
union all
select null, job, 'TOTAL BY JOB', sum(sal)
from emp
group by job
```

DEPTNO	JOB	CATEGORY	SAL
10	CLERK	TOTAL BY DEPT AND JOB	1300
10	MANAGER	TOTAL BY DEPT AND JOB	2450
10	PRESIDENT	TOTAL BY DEPT AND JOB	5000
20	CLERK	TOTAL BY DEPT AND JOB	1900
20	ANALYST	TOTAL BY DEPT AND JOB	6000
20	MANAGER	TOTAL BY DEPT AND JOB	2975
30	CLERK	TOTAL BY DEPT AND JOB	950
30	MANAGER	TOTAL BY DEPT AND JOB	2850
30	SALESMAN	TOTAL BY DEPT AND JOB	5600
	ANALYST	TOTAL BY JOB	6000
	CLERK	TOTAL BY JOB	4150
	MANAGER	TOTAL BY JOB	8275
	PRESIDENT	TOTAL BY JOB	5000
	SALESMAN	TOTAL BY JOB	5600

下一步，使用 UNION ALL，按 DEPTNO 计算所有工资的总和：

```
select deptno, job,
       'TOTAL BY DEPT AND JOB' as category,
       sum(sal) as sal
  from emp
 group by deptno, job
 union all
select null, job, 'TOTAL BY JOB', sum(sal)
  from emp
 group by job
 union all
select deptno, null, 'TOTAL BY DEPT', sum(sal)
  from emp
 group by deptno
```

DEPTNO	JOB	CATEGORY	SAL
10	CLERK	TOTAL BY DEPT AND JOB	1300
10	MANAGER	TOTAL BY DEPT AND JOB	2450
10	PRESIDENT	TOTAL BY DEPT AND JOB	5000
20	CLERK	TOTAL BY DEPT AND JOB	1900
20	ANALYST	TOTAL BY DEPT AND JOB	6000
20	MANAGER	TOTAL BY DEPT AND JOB	2975
30	CLERK	TOTAL BY DEPT AND JOB	950
30	MANAGER	TOTAL BY DEPT AND JOB	2850
30	SALESMAN	TOTAL BY DEPT AND JOB	5600
	ANALYST	TOTAL BY JOB	6000
	CLERK	TOTAL BY JOB	4150
	MANAGER	TOTAL BY JOB	8275
	PRESIDENT	TOTAL BY JOB	5000
	SALESMAN	TOTAL BY JOB	5600
10		TOTAL BY DEPT	8750
20		TOTAL BY DEPT	10875
30		TOTAL BY DEPT	9400

最后一步，使用 UNION ALL，计算表 EMP 中所有工资的总和：

```
select deptno, job,
       'TOTAL BY DEPT AND JOB' as category,
       sum(sal) as sal
  from emp
 group by deptno, job
 union all
select null, job, 'TOTAL BY JOB', sum(sal)
  from emp
 group by job
 union all
select deptno, null, 'TOTAL BY DEPT', sum(sal)
  from emp
 group by deptno
 union all
select null, null, 'GRAND TOTAL FOR TABLE', sum(sal)
  from emp
```

DEPTNO	JOB	CATEGORY	SAL
10	CLERK	TOTAL BY DEPT AND JOB	1300
10	MANAGER	TOTAL BY DEPT AND JOB	2450
10	PRESIDENT	TOTAL BY DEPT AND JOB	5000
20	CLERK	TOTAL BY DEPT AND JOB	1900
20	ANALYST	TOTAL BY DEPT AND JOB	6000
20	MANAGER	TOTAL BY DEPT AND JOB	2975
30	CLERK	TOTAL BY DEPT AND JOB	950
30	MANAGER	TOTAL BY DEPT AND JOB	2850
30	SALESMAN	TOTAL BY DEPT AND JOB	5600
	ANALYST	TOTAL BY JOB	6000

	CLERK	TOTAL BY JOB	4150
	MANAGER	TOTAL BY JOB	8275
	PRESIDENT	TOTAL BY JOB	5000
	SALESMAN	TOTAL BY JOB	5600
10		TOTAL BY DEPT	8750
20		TOTAL BY DEPT	10875
30		TOTAL BY DEPT	9400
		GRAND TOTAL FOR TABLE	29025

## 12.14 判别非小计的行

### 问题

前面曾经使用 GROUP BY 子句的 CUBE 扩展创建报表，而且需要采用一种方式区分哪些行是由常规 GROUP BY 子句生成的，哪些行是使用 CUBE 或 ROLLUP 的结果生成的。

下面的查询使用了 GROUP BY 子句的 CUBE 扩展，创建 EMP 表中工资的细目分类，其结果集如下：

DEPTNO	JOB	SAL
		29025
	CLERK	4150
	ANALYST	6000
	MANAGER	8275
	SALESMAN	5600
	PRESIDENT	5000
10		8750
10	CLERK	1300
10	MANAGER	2450
10	PRESIDENT	5000
20		10875
20	CLERK	1900
20	ANALYST	6000
20	MANAGER	2975
30		9400
30	CLERK	950
30	MANAGER	2850
30	SALESMAN	5600

这个报表包含按 DEPTNO 和 JOB（每个 DEPTNO 的每个 JOB）分组的所有工资总和、按 DEPTNO 分组的所有工资总和、按 JOB 分组的所有工资总和以及总计（表 EMP 中所有工资的总和）。现在需要明确地判别聚集的不同层，要判别每个聚集值属于何种类别（即 SAL 列中某给定值是按 DEPTNO 分组的和？按 JOB 分组的总和？抑或是总计？）。返回的结果集应如：

DEPTNO	JOB	SAL	DEPTNO_SUBTOTALS	JOB_SUBTOTALS
		29025	1	1
	CLERK	4150	1	0
	ANALYST	6000	1	0
	MANAGER	8275	1	0
	SALESMAN	5600	1	0
	PRESIDENT	5000	1	0
10		8750	0	1
10	CLERK	1300	0	0
10	MANAGER	2450	0	0
10	PRESIDENT	5000	0	0
20		10875	0	1

20	CLERK	1900	0	0
20	ANALYST	6000	0	0
20	MANAGER	2975	0	0
30		9400	0	1
30	CLERK	950	0	0
30	MANAGER	2850	0	0
30	SALESMAN	5600	0	0

解决方案

使用 GROUPING 函数，判别哪些值是由 CUBE 创建的小计、哪些值是由 ROLLUP 创建的小计，哪些值是总计值。下面给出了 DB2 和 Oracle 的例子：

```
1 select deptno, job, sum(sal) sal,
2      grouping(deptno) deptno_subtotals,
3      grouping(job) job_subtotals
4 from emp
5 group by cube(deptno, job)
```

SQL Server 解决方案与 DB2 和 Oracle 之间的唯一差别 CUBE/ROLLUP 子句的写法不同：

```
1 select deptno, job, sum(sal) sal,
2      grouping(deptno) deptno_subtotals,
3      grouping(job) job_subtotals
4 from emp
5 group by deptno, job with cube
```

本节主要讲述如何使用 CUBE 和 GROUPING 处理小计。到编写本书时，PostgreSQL 和 MySQL 既不支持 CUBE 也不支持 GROUPING。

讨论

如果 DEPTNO\_SUBTOTALS 为 1，则表示 SAL 值是由 CUBE 创建的按 DEPTNO 分组的小计；如果 JOB\_SUBTOTALS 为 1，则表示 SAL 值是由 CUBE 创建的按 JOB 分组的小计；如果 JOB\_SUBTOTALS 和 DEPTNO\_SUBTOTALS 都为 1，那么 SAL 值就表示由 CUBE 创建的所有工资的总计；当 DEPTNO\_SUBTOTALS 和 JOB\_SUBTOTALS 都为 0 时，表示这些行是由聚集按常规创建的（每个 DEPTNO/JOB 组合的 SAL 之和）。

12.15 使用 Case 表达式给行做标记

问题

把一列中的值（如 EMP 表的 JOB 列）映射成一系列“布尔”标记。例如，希望返回以下结果集：

ENAME	IS_CLERK	IS_SALES	IS_MGR	IS_ANALYST	IS_PREZ
KING	0	0	0	0	1
SCOTT	0	0	0	1	0
FORD	0	0	0	1	0
JONES	0	0	1	0	0
BLAKE	0	0	1	0	0
CLARK	0	0	1	0	0
ALLEN	0	1	0	0	0

WARD	0	1	0	0	0
MARTIN	0	1	0	0	0
TURNER	0	1	0	0	0
SMITH	1	0	0	0	0
MILLER	1	0	0	0	0
ADAMS	1	0	0	0	0
JAMES	1	0	0	0	0

这样的结果集可用于调试，它能够提供一个不同于其他典型结果集的数据视图。

## 解决方案

对每个雇员的 JOB 使用 CASE 表达式，并返回 1 或 0 表示他的 JOB。需要为每个可能的职位写一个 CASE 表达式，并创建一列：

```

1  select  ename,
2          case when job = 'CLERK'
3              then 1 else 0
4          end as is_clerk,
5          case when job = 'SALESMAN'
6              then 1 else 0
7          end as is_sales,
8          case when job = 'MANAGER'
9              then 1 else 0
10         end as is_mgr,
11         case when job = 'ANALYST'
12             then 1 else 0
13         end as is_analyst,
14         case when job = 'PRESIDENT'
15             then 1 else 0
16         end as is_prez
17  from emp
18  order by 2,3,4,5,6

```

## 讨论

该解决方案的代码几乎不用加以解释。如果不能理解它，那么只要把 JOB 添加到 SELECT 子句中就清楚了：

```

select  ename,
        job,
        case when job = 'CLERK'
            then 1 else 0
        end as is_clerk,
        case when job = 'SALESMAN'
            then 1 else 0
        end as is_sales,
        case when job = 'MANAGER'
            then 1 else 0
        end as is_mgr,
        case when job = 'ANALYST'
            then 1 else 0
        end as is_analyst,
        case when job = 'PRESIDENT'
            then 1 else 0
        end as is_prez
from emp
order by 2

```

ENAME	JOB	IS_CLERK	IS_SALES	IS_MGR	IS_ANALYST	IS_PREZ
SCOTT	ANALYST	0	0	0	1	0
FORD	ANALYST	0	0	0	1	0

SMITH	CLERK	1	0	0	0	0
ADAMS	CLERK	1	0	0	0	0
MILLER	CLERK	1	0	0	0	0
JAMES	CLERK	1	0	0	0	0
JONES	MANAGER	0	0	1	0	0
CLARK	MANAGER	0	0	1	0	0
BLAKE	MANAGER	0	0	1	0	0
KING	PRESIDENT	0	0	0	0	1
ALLEN	SALESMAN	0	1	0	0	0
MARTIN	SALESMAN	0	1	0	0	0
TURNER	SALESMAN	0	1	0	0	0
WARD	SALESMAN	0	1	0	0	0

## 12.16 创建稀疏矩阵

### 问题

创建一个稀疏矩阵，例如，下面的例子就是由表EMP的DEPTNO列和JOB列变换来的：

D10	D20	D30	CLERKS	MGRS	PREZ	ANALS	SALES
	SMITH		SMITH				
		ALLEN WARD					ALLEN WARD
	JONES			JONES			
		MARTIN BLAKE					MARTIN
CLARK				BLAKE CLARK			
	SCOTT					SCOTT	
KING					KING		
	ADAMS	TURNER	ADAMS				TURNER
		JAMES	JAMES				
	FORD					FORD	
MILLER			MILLER				

### 解决方案

使用CASE表达式创建稀疏的行到列的变换：

```

1  select case deptno when 10 then ename end as d10,
2         case deptno when 20 then ename end as d20,
3         case deptno when 30 then ename end as d30,
4         case job when 'CLERK' then ename end as clerks,
5         case job when 'MANAGER' then ename end as mgrs,
6         case job when 'PRESIDENT' then ename end as prez,
7         case job when 'ANALYST' then ename end as anals,
8         case job when 'SALESMAN' then ename end as sales
9  from emp

```

### 讨论

把DEPTNO和JOB行变换为列，只需使用CASE表达式，判断由这些行返回的可能值。这就是对它的所有解释。另外，如果想使报表变“稠密”，并去除一些NULL行，就需要找到分组的依据。例如，使用窗口函数ROW\_NUMBER OVER，为每个DEPTNO中的每个员工分等级，然后使用聚集函数MAX去掉一些NULL：

```

select max(case deptno when 10 then ename end) d10,
       max(case deptno when 20 then ename end) d20,

```

```

max(case deptno when 30 then ename end) d30,
max(case job when 'CLERK' then ename end) clerks,
max(case job when 'MANAGER' then ename end) mgrs,
max(case job when 'PRESIDENT' then ename end) prez,
max(case job when 'ANALYST' then ename end) anals,
max(case job when 'SALESMAN' then ename end) sales
from (
select deptno, job, ename,
row_number()over(partition by deptno order by empno) rn
from emp
) x
group by rn

```

D10	D20	D30	CLERKS	MGRS	PREZ	ANALS	SALES
CLARK	SMITH	ALLEN	SMITH	CLARK			ALLEN
KING	JONES	WARD		JONES	KING		WARD
MILLER	SCOTT	MARTIN	MILLER			SCOTT	MARTIN
	ADAMS	BLAKE	ADAMS	BLAKE			
	FORD	TURNER				FORD	TURNER
		JAMES	JAMES				

## 12.17 按时间单位给行分组

### 问题

按某个时间间隔计算数据的和。例如，有一个事务处理日志，想求得每 5 秒钟内的总事务数。表 TRX\_LOG 中的行如下所示：

```

select trx_id,
       trx_date,
       trx_cnt
from   trx_log

```

TRX_ID	TRX_DATE	TRX_CNT
1	28-JUL-2005 19: 03: 07	44
2	28-JUL-2005 19: 03: 08	18
3	28-JUL-2005 19: 03: 09	23
4	28-JUL-2005 19: 03: 10	29
5	28-JUL-2005 19: 03: 11	27
6	28-JUL-2005 19: 03: 12	45
7	28-JUL-2005 19: 03: 13	45
8	28-JUL-2005 19: 03: 14	32
9	28-JUL-2005 19: 03: 15	41
10	28-JUL-2005 19: 03: 16	15
11	28-JUL-2005 19: 03: 17	24
12	28-JUL-2005 19: 03: 18	47
13	28-JUL-2005 19: 03: 19	37
14	28-JUL-2005 19: 03: 20	48
15	28-JUL-2005 19: 03: 21	46
16	28-JUL-2005 19: 03: 22	44
17	28-JUL-2005 19: 03: 23	36
18	28-JUL-2005 19: 03: 24	41
19	28-JUL-2005 19: 03: 25	33
20	28-JUL-2005 19: 03: 26	19

要返回如下结果集：

GRP	TRX_START	TRX_END	TOTAL
1	28-JUL-2005 19: 03: 07	28-JUL-2005 19: 03: 11	141
2	28-JUL-2005 19: 03: 12	28-JUL-2005 19: 03: 16	178
3	28-JUL-2005 19: 03: 17	28-JUL-2005 19: 03: 21	202



解决方案

把所有项分组为每5行1桶。有很多方式可实现这种逻辑分组，本节采用了用TRX\_ID除以5的技巧，请参阅本章第12.7节。

一旦创建了“组”，就可以使用聚集函数MIN、MAX和SUM求起始时间、结束时间及每个“组”的事务处理总数（SQL Server用户应该使用CEILING，而不是使用CEIL）：

```
1 select ceil(trx_id/5.0) as grp,
2      min(trx_date)      as trx_start,
3      max(trx_date)      as trx_end,
4      sum(trx_cnt)        as total
5 from   trx_log
6 group by ceil(trx_id/5.0)
```

讨论

第一步，也是整个解决方案的关键，先对行进行逻辑分组，再除以5，然后取大于商的最小整数，就可以创建逻辑组。例如：

```
select  trx_id,
        trx_date,
        trx_cnt,
        trx_id/5.0      as val,
        ceil(trx_id/5.0) as grp
from    trx_log
```

TRX_ID	TRX_DATE	TRX_CNT	VAL	GRP
1	28-JUL-2005 19: 03: 07	44	.20	1
2	28-JUL-2005 19: 03: 08	18	.40	1
3	28-JUL-2005 19: 03: 09	23	.60	1
4	28-JUL-2005 19: 03: 10	29	.80	1
5	28-JUL-2005 19: 03: 11	27	1.00	1
6	28-JUL-2005 19: 03: 12	45	1.20	2
7	28-JUL-2005 19: 03: 13	45	1.40	2
8	28-JUL-2005 19: 03: 14	32	1.60	2
9	28-JUL-2005 19: 03: 15	41	1.80	2
10	28-JUL-2005 19: 03: 16	15	2.00	2
11	28-JUL-2005 19: 03: 17	24	2.20	3
12	28-JUL-2005 19: 03: 18	47	2.40	3
13	28-JUL-2005 19: 03: 19	37	2.60	3
14	28-JUL-2005 19: 03: 20	48	2.80	3
15	28-JUL-2005 19: 03: 21	46	3.00	3
16	28-JUL-2005 19: 03: 22	44	3.20	4
17	28-JUL-2005 19: 03: 23	36	3.40	4
18	28-JUL-2005 19: 03: 24	41	3.60	4
19	28-JUL-2005 19: 03: 25	33	3.80	4
20	28-JUL-2005 19: 03: 26	19	4.00	4

最后一步，使用合适的聚集函数，按每5秒的时间间隔求事务处理总数以及每个事务处理的起始时间和结束时间：

```
select  ceil(trx_id/5.0) as grp,
        min(trx_date)    as trx_start,
        max(trx_date)    as trx_end,
        sum(trx_cnt)      as total
```

```

from trx_log
group by ceil(trx_id/5.0)

```

GRP	TRX_START	TRX_END	TOTAL
1	28-JUL-2005 19: 03: 07	28-JUL-2005 19: 03: 11	141
2	28-JUL-2005 19: 03: 12	28-JUL-2005 19: 03: 16	178
3	28-JUL-2005 19: 03: 17	28-JUL-2005 19: 03: 21	202
4	28-JUL-2005 19: 03: 22	28-JUL-2005 19: 03: 26	173

如果实际数据与上述例子不同（也许每行不包含 ID），则总可以先用 TRX\_DATE 行的秒数除以 5 进行类似的分组。然后，引入每个 TRX\_DATE 的小时值，并按实际小时值及逻辑“分组”GRP 进行分组。下面给出了采用这种技巧的一个例子（使用 Oracle 的 TO\_CHAR 和 TO\_NUMBER 函数，在自己的平台中应使用恰当的日期和字符格式函数）：

```

select trx_date, trx_cnt,
       to_number(to_char(trx_date, 'hh24')) hr,
       ceil(to_number(to_char(trx_date-1/24/60/60, 'miss'))/5.0) grp
from trx_log

```

TRX_DATE	TRX_CNT	HR	GRP
28-JUL-2005 19: 03: 07	44	19	62
28-JUL-2005 19: 03: 08	18	19	62
28-JUL-2005 19: 03: 09	23	19	62
28-JUL-2005 19: 03: 10	29	19	62
28-JUL-2005 19: 03: 11	27	19	62
28-JUL-2005 19: 03: 12	45	19	63
28-JUL-2005 19: 03: 13	45	19	63
28-JUL-2005 19: 03: 14	32	19	63
28-JUL-2005 19: 03: 15	41	19	63
28-JUL-2005 19: 03: 16	15	19	63
28-JUL-2005 19: 03: 17	24	19	64
28-JUL-2005 19: 03: 18	47	19	64
28-JUL-2005 19: 03: 19	37	19	64
28-JUL-2005 19: 03: 20	48	19	64
28-JUL-2005 19: 03: 21	46	19	64
28-JUL-2005 19: 03: 22	44	19	65
28-JUL-2005 19: 03: 23	36	19	65
28-JUL-2005 19: 03: 24	41	19	65
28-JUL-2005 19: 03: 25	33	19	65
28-JUL-2005 19: 03: 26	19	19	65

这里，关键是按 5 秒间隔分组，并不关心 GRP 的实际值。自此，可以按照初始解决方案中介绍的方式使用聚集函数：

```

select hr, grp, sum(trx_cnt) total
from (
select trx_date, trx_cnt,
       to_number(to_char(trx_date, 'hh24')) hr,
       ceil(to_number(to_char(trx_date-1/24/60/60, 'miss'))/5.0) grp
from trx_log
) x
group by hr, grp

```

HR	GRP	TOTAL
19	62	141
19	63	178
19	64	202
19	65	173

如果事务处理日志跨越了几小时，那么在分组中包括小时单位是很有用的。在 DB2 和

Oracle 中，也可以使用窗口函数 SUM OVER 产生同样的结果。下面的查询将返回 TRX\_LOG 的所有行（各行包含按逻辑“组”的 TRX\_CNT 的累计和），以及“组”中各行 TRX\_CNT 的合计：

```
select trx_id, trx_date, trx_cnt,
       sum(trx_cnt)over(partition by ceil(trx_id/5.0)
                        order by trx_date
                        range between unbounded preceding
                               and current row) runing_total,
       sum(trx_cnt)over(partition by ceil(trx_id/5.0)) total,
       case when mod(trx_id,5.0) = 0 then 'X' end grp_end
from   trx_log
```

TRX_ID	TRX_DATE	TRX_CNT	RUNING_TOTAL	TOTAL	GRP_END
1	28-JUL-2005 19: 03: 07	44	44	141	
2	28-JUL-2005 19: 03: 08	18	62	141	
3	28-JUL-2005 19: 03: 09	23	85	141	
4	28-JUL-2005 19: 03: 10	29	114	141	
5	28-JUL-2005 19: 03: 11	27	141	141 X	
6	28-JUL-2005 19: 03: 12	45	45	178	
7	28-JUL-2005 19: 03: 13	45	90	178	
8	28-JUL-2005 19: 03: 14	32	122	178	
9	28-JUL-2005 19: 03: 15	41	163	178	
10	28-JUL-2005 19: 03: 16	15	178	178 X	
11	28-JUL-2005 19: 03: 17	24	24	202	
12	28-JUL-2005 19: 03: 18	47	71	202	
13	28-JUL-2005 19: 03: 19	37	108	202	
14	28-JUL-2005 19: 03: 20	48	156	202	
15	28-JUL-2005 19: 03: 21	46	202	202 X	
16	28-JUL-2005 19: 03: 22	44	44	173	
17	28-JUL-2005 19: 03: 23	36	80	173	
18	28-JUL-2005 19: 03: 24	41	121	173	
19	28-JUL-2005 19: 03: 25	33	154	173	
20	28-JUL-2005 19: 03: 26	19	173	173 X	

## 12.18 对不同组 / 分区同时实现聚集问题

同时按不同维进行聚集。例如，要返回这样的结果集：列出每个员工的名字、他所在的部门、该部门的员工数（包括他自己）、与他有同样职位的员工数（也包括他自己）以及 EMP 表中的员工总数。其结果集应该如下所示：

ENAME	DEPTNO	DEPTNO_CNT	JOB	JOB_CNT	TOTAL
MILLER	10	3	CLERK	4	14
CLARK	10	3	MANAGER	3	14
KING	10	3	PRESIDENT	1	14
SCOTT	20	5	ANALYST	2	14
FORD	20	5	ANALYST	2	14
SMITH	20	5	CLERK	4	14
JONES	20	5	MANAGER	3	14
ADAMS	20	5	CLERK	4	14
JAMES	30	6	CLERK	4	14
MARTIN	30	6	SALESMAN	4	14
TURNER	30	6	SALESMAN	4	14
WARD	30	6	SALESMAN	4	14
ALLEN	30	6	SALESMAN	4	14
BLAKE	30	6	MANAGER	3	14

## 解决方案

窗口函数使这个问题相当容易解决。如果不能使用窗口函数，也可以使用标量子查询。

### DB2、Oracle 和 SQL Server

使用 COUNT OVER 窗口函数进行聚集，分别指定不同的分区或分组数据：

```
select ename,
       deptno,
       count(*)over(partition by deptno) deptno_cnt,
       job,
       count(*)over(partition by job) job_cnt,
       count(*)over() total
from emp
```

### PostgreSQL 和 MySQL

在 SELECT 列表中使用标量子查询，就可以针对不同行组完成聚集计算操作：

```
1 select e.ename,
2       e.deptno,
3       (select count(*) from emp d
4        where d.deptno = e.deptno) as deptno_cnt,
5       job,
6       (select count(*) from emp d
7        where d.job = e.job) as job_cnt,
8       (select count(*) from emp) as total
9 from emp e
```

## 讨论

### DB2、Oracle 和 SQL Server

这个例子确实展现出了窗口函数的强大和方便。只要指定不同分区和分组，就可以创建非常详尽的报表，而无需进行反反复复的自联接，也不必在 SELECT 列表中编写乏味而且可能非常低效的子查询，所有工作都由窗口函数 COUNT OVER 完成。要理解输出结果，咀嚼一下每个 COUNT 操作的 OVER 子句即可：

```
count(*)over(partition by deptno)
count(*)over(partition by job)
count(*)over()
```

记住 OVER 子句的主要部分是：由 PARTITION BY 指定的分区和由 ORDER BY 指定的框架或窗口。先看第一个 COUNT，它按 DEPTNO 分区。表 EMP 中的行将按 DEPTNO 分组，并且会对每组的行进行 COUNT 操作，由于没有指定框架或窗口子句（没有 ORDER BY），所以会计算该组中的所有行。PARTITION BY 子句将找出所有不重复的 DEPTNO 值，然后 COUNT 函数会计算每个值的行数。在本例的 COUNT(\*)OVER(PARTITION BY DEPTNO) 中，PARTITION BY 子句找出，分区或组有 10、20 和 30 三个值。

第二个 COUNT 也进行同样的操作，只不过它是按 JOB 分区的。最后一个 COUNT 没有按任何值分区，它仅仅引入了一个空括号。空括号意味着“整个表”。因此，前两个 COUNT 是基于已定义的组或分区完成聚集的，而最后一个 COUNT 计算了表 EMP 中的所有行。

**警告：**记住，在 WHERE 子句后面使用窗口函数。如果想以某种方式筛选结果集，例如，去掉 DEPTNO 10 中的所有员工，那么 TOTAL 值将为 14，筛选后的值为 11。要想在计算窗口函数之后筛选结果，必须使窗口查询成为内联视图，然后从该视图中筛选结果。

## PostgreSQL 和 MySQL

对于主查询返回的每一行（EMP E 中的行），可以在 SELECT 列表中使用多个标量子查询，完成对每个 DEPTNO 和 JOB 的不同计算。要得到 TOTAL，简单地使用另一个标量子查询，获取表 EMP 中所有员工的数目。

## 12.19 对移动范围的值进行聚集

### 问题

要计算移动聚集，例如，求表 EMP 中工资的移动和，要从第一个员工的 HIREDATE 开始，计算任何 90 天内的总和，以观察从雇用第一个员工至雇用最后一个员工期间 90 天开销的波动情况。应返回以下结果集：

HIREDATE	SAL	SPENDING_PATTERN
17-DEC-1980	800	800
20-FEB-1981	1600	2400
22-FEB-1981	1250	3650
02-APR-1981	2975	5825
01-MAY-1981	2850	8675
09-JUN-1981	2450	8275
08-SEP-1981	1500	1500
28-SEP-1981	1250	2750
17-NOV-1981	5000	7750
03-DEC-1981	950	11700
03-DEC-1981	3000	11700
23-JAN-1982	1300	10250
09-DEC-1982	3000	3000
12-JAN-1983	1100	4100

### 解决方案

如果 RDBMS 支持在窗口函数的框架或窗口子句中指定可移动窗口，这个问题就很容易解决。关键是需要窗口函数中按 HIREDATE 排序，然后指定一个从雇用第一个员工开始的 90 天窗口。和的计算方法是当前员工的 HIREDATE 向前 90 天内（当前员工也包括在内）聘用的员工的工资。如果无法使用这样的窗口函数，则可以使用标量子查询，但其解决方案可能会更复杂。

## DB2 和 Oracle

对于 DB2 和 Oracle，可使用窗口函数 SUM OVER，并按 HIREDATE 排序。在窗口或“框架”子句中指定 90 天的范围，计算之前 90 天前聘用的所有员工的工资之和。因为 DB2

不允许在窗口函数的 ORDER BY 子句中指定 HIREDATE (下面的第 3 行代码), 可以按 DAYS(HIREDATE)排序:

```
1 select hiredate,
2       sal,
3       sum(sal)over(order by days(hiredate)
4                   range between 90 preceding
5                   and current row) spending_pattern
6 from emp e
```

Oracle 解决方案比 DB2 更简单易懂, 这是由于 Oracle 允许窗口函数按 datetime 类型排序:

```
1 select hiredate,
2       sal,
3       sum(sal)over(order by hiredate
4                   range between 90 preceding
5                   and current row) spending_pattern
6 from emp e
```

## MySQL、PostgreSQL 和 SQL Server

使用标量子查询, 计算某日期之前聘用的所有员工的工资和:

```
1 select e.hiredate,
2       e.sal,
3       (select sum(sal) from emp d
4        where d.hiredate between e.hiredate-90
5              and e.hiredate) as spending_pattern
6 from emp e
7 order by 1
```

## 讨论

### DB2 和 Oracle

DB2 和 Oracle 具有同样的解决方案。两个解决方案的唯一差别是窗口函数的 ORDER BY 子句指定 HIREDATE 的方式不同。至编写本书时, 如果使用数字值设置窗口范围, DB2 不允许在 ORDER BY 子句中使用 DATE 类型值。(例如, RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW 允许按日期排序, 但 RANGE BETWEEN 90 PRECEDING AND CURRENT ROW 不允许这样做)。

要理解该解决方案查询的功能, 只需要了解窗口子句的功能。定义的窗口会按 HIREDATE 给所有员工的工资排序, 然后, 函数再求和。这里并不是对所有工资求和, 而是按以下过程处理:

1. 计算第一个员工的工资。由于没有其他员工是在该员工之前聘用的, 所以这时的和就是第一个员工的工资。
2. 计算下一个员工 (按 HIREDATE 顺序) 的工资。这个员工以及聘用他之前 90 天内聘用的其他员工的工资都计入移动和之中。

第一个员工的 HIREDATE 是 1980 年 12 月 17 日, 下一个员工的 HIREDATE 是 1981 年 2



月20日,第二个员工是在聘用第一个员工后90天之内聘用的,因此第二个员工的移动和值是2400(1600+800)。如果不理解 SPENDING\_PATTERN 中的值是从哪儿来的,那么可以检验下列查询及其结果集:

```
select distinct
  dense_rank()over(order by e.hiredate) window,
  e.hiredate current_hiredate,
  d.hiredate hiredate_within_90_days,
  d.sal sals_used_for_sum
from emp e,
     emp d
where d.hiredate between e.hiredate-90 and e.hiredate
```

WINDOW	CURRENT_HIREDATE	HIREDATE_WITHIN_90_DAYS	SALS_USED_FOR_SUM
1	17-DEC-1980	17-DEC-1980	800
2	20-FEB-1981	17-DEC-1980	800
2	20-FEB-1981	20-FEB-1981	1600
3	22-FEB-1981	17-DEC-1980	800
3	22-FEB-1981	20-FEB-1981	1600
3	22-FEB-1981	22-FEB-1981	1250
4	02-APR-1981	20-FEB-1981	1600
4	02-APR-1981	22-FEB-1981	1250
4	02-APR-1981	02-APR-1981	2975
5	01-MAY-1981	20-FEB-1981	1600
5	01-MAY-1981	22-FEB-1981	1250
5	01-MAY-1981	02-APR-1981	2975
5	01-MAY-1981	01-MAY-1981	2850
6	09-JUN-1981	02-APR-1981	2975
6	09-JUN-1981	01-MAY-1981	2850
6	09-JUN-1981	09-JUN-1981	2450
7	08-SEP-1981	08-SEP-1981	1500
8	28-SEP-1981	08-SEP-1981	1500
8	28-SEP-1981	28-SEP-1981	1250
9	17-NOV-1981	08-SEP-1981	1500
9	17-NOV-1981	28-SEP-1981	1250
9	17-NOV-1981	17-NOV-1981	5000
10	03-DEC-1981	08-SEP-1981	1500
10	03-DEC-1981	28-SEP-1981	1250
10	03-DEC-1981	17-NOV-1981	5000
10	03-DEC-1981	03-DEC-1981	950
10	03-DEC-1981	03-DEC-1981	3000
11	23-JAN-1982	17-NOV-1981	5000
11	23-JAN-1982	03-DEC-1981	950
11	23-JAN-1982	03-DEC-1981	3000
11	23-JAN-1982	23-JAN-1982	1300
12	09-DEC-1982	09-DEC-1982	3000
13	12-JAN-1983	09-DEC-1982	3000
13	12-JAN-1983	12-JAN-1983	1100

如果观察一下 WINDOW 列就会发现,每次求和只将具有相同 WINDOW 值的行相加。例如,WINDOW 3,该窗口中参与求和的工资分别为800、1600和1250,总共为3650。如果查看“问题”一节中的最终结果集,会发现1981年2月22日(WINDOW 3)的 SPENDING\_PATTERN 是3650。要验证上面的自联接为相应窗口选择工资是否正确,只需计算 SALS\_USED\_FOR\_SUM 值的和,并按 CURRENT\_DATE 分组。其结果应该与“问题”一节中的结果集相同(1981年12月3日的重复行已经筛选掉了):

```
select current_hiredate,
       sum(sals_used_for_sum) spending_pattern
from (
```

```

select distinct
  dense_rank()over(order by e.hiredate) window,
  e.hiredate current_hiredate,
  d.hiredate hiredate_within_90_days,
  d.sal sals_used_for_sum
  from emp e,
       emp d
  where d.hiredate between e.hiredate-90 and e.hiredate
        ) x
  group by current_hiredate

```

CURRENT_HIREDATE	SPENDING_PATTERN
17-DEC-1980	800
20-FEB-1981	2400
22-FEB-1981	3650
02-APR-1981	5825
01-MAY-1981	8675
09-JUN-1981	8275
08-SEP-1981	1500
28-SEP-1981	2750
17-NOV-1981	7750
03-DEC-1981	11700
23-JAN-1982	10250
09-DEC-1982	3000
12-JAN-1983	4100

## MySQL, PostgreSQL 和 SQL Server

在这个解决方案中，关键是使用标量子查询（自联接也能实现），并使用聚集函数 SUM 基于 HIREDATE 计算 90 天的和。如果不明白这是如何实现的，只要把该解决方案转换为使用自联接，并检验计算结果中包含了哪些行。参阅下面的结果集，它返回的结果与解决方案中的结果相同：

```

select e.hiredate,
       e.sal,
       sum(d.sal) as spending_pattern
  from emp e, emp d
  where d.hiredate
        between e.hiredate-90 and e.hiredate
  group by e.hiredate,e.sal
  order by 1

```

HIREDATE	SAL	SPENDING_PATTERN
17-DEC-1980	800	800
20-FEB-1981	1600	2400
22-FEB-1981	1250	3650
02-APR-1981	2975	5825
01-MAY-1981	2850	8675
09-JUN-1981	2450	8275
08-SEP-1981	1500	1500
28-SEP-1981	1250	2750
17-NOV-1981	5000	7750
03-DEC-1981	950	11700
03-DEC-1981	3000	11700
23-JAN-1982	1300	10250
09-DEC-1982	3000	3000
12-JAN-1983	1100	4100

如果还是不明白，则先去掉聚集，从笛卡儿积开始。第一步，使用表 EMP 生成笛卡儿积，这样，每个 HIREDATE 都能与其他的 HIREDATE 相比较 [下面只显示了结果集的片段，因为 EMP 的笛卡儿积会返回 196 行 (14X14)]：



```
select e.hiredate,
       e.sal,
       d.sal,
       d.hiredate
from emp e, emp d
```

HIREDATE	SAL	SAL	HIREDATE
17-DEC-1980	800	800	17-DEC-1980
17-DEC-1980	800	1600	20-FEB-1981
17-DEC-1980	800	1250	22-FEB-1981
17-DEC-1980	800	2975	02-APR-1981
17-DEC-1980	800	1250	28-SEP-1981
17-DEC-1980	800	2850	01-MAY-1981
17-DEC-1980	800	2450	09-JUN-1981
17-DEC-1980	800	3000	09-DEC-1982
17-DEC-1980	800	5000	17-NOV-1981
17-DEC-1980	800	1500	08-SEP-1981
17-DEC-1980	800	1100	12-JAN-1983
17-DEC-1980	800	950	03-DEC-1981
17-DEC-1980	800	3000	03-DEC-1981
17-DEC-1980	800	1300	23-JAN-1982
20-FEB-1981	1600	800	17-DEC-1980
20-FEB-1981	1600	1600	20-FEB-1981
20-FEB-1981	1600	1250	22-FEB-1981
20-FEB-1981	1600	2975	02-APR-1981
20-FEB-1981	1600	1250	28-SEP-1981
20-FEB-1981	1600	2850	01-MAY-1981
20-FEB-1981	1600	2450	09-JUN-1981
20-FEB-1981	1600	3000	09-DEC-1982
20-FEB-1981	1600	5000	17-NOV-1981
20-FEB-1981	1600	1500	08-SEP-1981
20-FEB-1981	1600	1100	12-JAN-1983
20-FEB-1981	1600	950	03-DEC-1981
20-FEB-1981	1600	3000	03-DEC-1981
20-FEB-1981	1600	1300	23-JAN-1982

检验一下上面的结果集就会注意到，除1980年12月17日之外，没有其他HIREDATE在它之前90天之内或跟它相同，所以该行的和值应该为800。再检查下一个HIREDATE，即1981年2月20日，就会注意到，有一个HIREDATE落入了90天窗口内（之前90天以内），它就是1981年12月17日。如果把1981年12月17日的SAL与1981年2月20日的SAL相加（因为要找的就是等于该HIREDATE或在它之前90天内的其他HIREDATE），则得到2400，它就是该HIREDATE的最终结果。

现在应该明白其操作过程了。在WHERE子句中加上筛选条件，就可以返回每个HIREDATE，以及那些等于它或在它之前90天之内的HIREDATE：

```
select e.hiredate,
       e.sal,
       d.sal sal_to_sum,
       d.hiredate within_90_days
from emp e, emp d
where d.hiredate
      between e.hiredate-90 and e.hiredate
order by 1
```

HIREDATE	SAL	SAL_TO_SUM	WITHIN_90_DAYS
17-DEC-1980	800	800	17-DEC-1980
20-FEB-1981	1600	800	17-DEC-1980
20-FEB-1981	1600	1600	20-FEB-1981

22-FEB-1981	1250	800	17-DEC-1980
22-FEB-1981	1250	1600	20-FEB-1981
22-FEB-1981	1250	1250	22-FEB-1981
02-APR-1981	2975	1600	20-FEB-1981
02-APR-1981	2975	1250	22-FEB-1981
02-APR-1981	2975	2975	02-APR-1981
01-MAY-1981	2850	1600	20-FEB-1981
01-MAY-1981	2850	1250	22-FEB-1981
01-MAY-1981	2850	2975	02-APR-1981
01-MAY-1981	2850	2850	01-MAY-1981
09-JUN-1981	2450	2975	02-APR-1981
09-JUN-1981	2450	2850	01-MAY-1981
09-JUN-1981	2450	2450	09-JUN-1981
08-SEP-1981	1500	1500	08-SEP-1981
28-SEP-1981	1250	1500	08-SEP-1981
28-SEP-1981	1250	1250	28-SEP-1981
17-NOV-1981	5000	1500	08-SEP-1981
17-NOV-1981	5000	1250	28-SEP-1981
17-NOV-1981	5000	5000	17-NOV-1981
03-DEC-1981	950	1500	08-SEP-1981
03-DEC-1981	950	1250	28-SEP-1981
03-DEC-1981	950	5000	17-NOV-1981
03-DEC-1981	950	950	03-DEC-1981
03-DEC-1981	950	3000	03-DEC-1981
03-DEC-1981	3000	1500	08-SEP-1981
03-DEC-1981	3000	1250	28-SEP-1981
03-DEC-1981	3000	5000	17-NOV-1981
03-DEC-1981	3000	950	03-DEC-1981
03-DEC-1981	3000	3000	03-DEC-1981
23-JAN-1982	1300	5000	17-NOV-1981
23-JAN-1982	1300	950	03-DEC-1981
23-JAN-1982	1300	3000	03-DEC-1981
23-JAN-1982	1300	1300	23-JAN-1982
09-DEC-1982	3000	3000	09-DEC-1982
12-JAN-1983	1100	3000	09-DEC-1982
12-JAN-1983	1100	1100	12-JAN-1983

知道了求和移动窗口中包含了哪些SAL之后，使用聚集函数SUM，产生一个表达更清晰的结果集：

```
select e.hiredate,
       e.sal,
       sum(d.sal) as spending_pattern
  from emp e, emp d
 where d.hiredate
        between e.hiredate-90 and e.hiredate
 group by e.hiredate,e.sal
 order by 1
```

如果把上面查询的结果集与下面查询（这是最初的解决方案）的结果集相比较，就会发现它们是一样的：

```
select e.hiredate,
       e.sal,
       (select sum(sal) from emp d
        where d.hiredate between e.hiredate-90
                          and e.hiredate) as spending_pattern
  from emp e
 order by 1
```

HIREDATE	SAL	SPENDING_PATTERN
17-DEC-1980	800	800
20-FEB-1981	1600	2400
22-FEB-1981	1250	3650

02-APR-1981	2975	5825
01-MAY-1981	2850	8675
09-JUN-1981	2450	8275
08-SEP-1981	1500	1500
28-SEP-1981	1250	2750
17-NOV-1981	5000	7750
03-DEC-1981	950	11700
03-DEC-1981	3000	11700
23-JAN-1982	1300	10250
09-DEC-1982	3000	3000
12-JAN-1983	1100	4100

12.20 转置带小计的结果集问题

要创建一个包含小计的报表，然后对结果做转置变换，使报表更易读。例如，要求创建一个报表，它显示每个部门、部门经理，以及这些经理手下员工的总工资。另外，还要返回两个小计：每个部门中各经理手下员工工资的总和、结果集中的所有工资总和（部门小计的总和）。目前有如下报表：

DEPTNO	MGR	SAL
10	7782	1300
10	7839	2450
10		3750
20	7566	6000
20	7788	1100
20	7839	2975
20	7902	800
20		10875
30	7698	6550
30	7839	2850
30		9400
		24025

要提供一个更易读的报表，而且希望把上面的结果集转换为下列格式，它会使报表的意义更为清晰：

MGR	DEPT10	DEPT20	DEPT30	TOTAL
7566	0	6000	0	
7698	0	0	6550	
7782	1300	0	0	
7788	0	1100	0	
7839	2450	2975	2850	
7902	0	800	0	
	3750	10875	9400	24025

解决方案

首先，使用 GROUP BY 的 ROLLUP 扩展生成小计，然后进行经典的转置变换（聚集及 CASE 表达式），以创建报表所需要的列。使用 GROUPING 函数，很容易确定哪些值是小计（即，正常情况下没有，加了 ROLLUP 才有的行）。根据 RDBMS 给 NULL 值排序的方式，可能需要在解决方案中添加 ORDER BY，使结果更像上面的目标结果集。

## DB2 和 Oracle

使用 GROUP BY 的 ROLLUP 扩展，然后使用 CASE 表达式，把数据格式设置为更易读的报表：

```
1 select mgr,
2       sum(case deptno when 10 then sal else 0 end) dept10,
3       sum(case deptno when 20 then sal else 0 end) dept20,
4       sum(case deptno when 30 then sal else 0 end) dept30,
5       sum(case flag   when '11' then sal else null end) total
6   from (
7 select deptno,mgr,sum(sal) sal,
8       cast(grouping(deptno) as char(1))||
9       cast(grouping(mgr)   as char(1)) flag
10  from emp
11 where mgr is not null
12 group by rollup(deptno,mgr)
13        ) x
14 group by mgr
```

## SQL Server

使用 GROUP BY 的 ROLLUP 扩展，然后使用 CASE 表达式，把数据设置为更易读的报表格式：

```
1 select mgr,
2       sum(case deptno when 10 then sal else 0 end) dept10,
3       sum(case deptno when 20 then sal else 0 end) dept20,
4       sum(case deptno when 30 then sal else 0 end) dept30,
5       sum(case flag   when '11' then sal else null end) total
6   from (
7 select deptno,mgr,sum(sal) sal,
8       cast(grouping(deptno) as char(1))+
9       cast(grouping(mgr)   as char(1)) flag
10  from emp
11 where mgr is not null
12 group by deptno,mgr with rollup
13        ) x
14 group by mgr
```

## MySQL 和 PostgreSQL

两个 RDBMS 都不支持 GROUPING 函数。

## 讨论

上面的解决方案都相同，仅仅字符串连接及指定 GROUPING 的方式不同。由于这些解决方案相似，所以下面的讨论仅针对 SQL Server 解决方案，以突出中间结果集（本讨论经相应修改后也适用于 DB2 和 Oracle）。

第一步，生成结果集，对于每个 MGR，分别计算他在各 DEPTNO 手下员工的 SAL 之和。其方法是求特定部门中某特定经理手下的员工收入。例如，下面的查询可以同时给出 DEPTNO 10 和 DEPTNO 30 中 KING 手下员工的工资：

```
select deptno,mgr,sum(sal) sal
from emp
```

```

where mgr is not null
group by mgr,deptno
order by 1,2

```

DEPTNO	MGR	SAL
10	7782	1300
10	7839	2450
20	7566	6000
20	7788	1100
20	7839	2975
20	7902	800
30	7698	6550
30	7839	2850

下一步，使用 GROUP BY 的 ROLLUP 扩展，为每个 DEPTNO 的所有员工创建小计：

```

select deptno,mgr,sum(sal) sal
  from emp
 where mgr is not null
group by deptno,mgr with rollup

```

DEPTNO	MGR	SAL
10	7782	1300
10	7839	2450
10		3750
20	7566	6000
20	7788	1100
20	7839	2975
20	7902	800
20		10875
30	7698	6550
30	7839	2850
30		9400
		24025

对已创建的小计，需要确定哪些值真正的小计（由 ROLLUP 创建的），哪些值是常规 GROUP BY 的结果。使用 GROUPING 函数创建位图，以便于区分常规聚集值与小计值：

```

select deptno,mgr,sum(sal) sal,
       cast(grouping(deptno) as char(1))+
       cast(grouping(mgr) as char(1)) flag
  from emp
 where mgr is not null
group by deptno,mgr with rollup

```

DEPTNO	MGR	SAL	FLAG
10	7782	1300	00
10	7839	2450	00
10		3750	01
20	7566	6000	00
20	7788	1100	00
20	7839	2975	00
20	7902	800	00
20		10875	01
30	7698	6550	00
30	7839	2850	00
30		9400	01
		24025	11

可能不怎么明显，FLAG 值为 00 的行是常规聚集的结果；而 FLAG 值为 01 的行是按

DEPTNO 对 SAL 进行 ROLLUP 聚集的结果 (因为在 ROLLUP 中先列出了 DEPTNO; 如果改变顺序, 例如, “GROUP BY MGR, DEPTNO WITH ROLLUP”, 就会得到另一种结果); FLAG 值为 11 的行是对所有行的 SAL 进行 ROLLUP 聚集的结果。

至此, 只需使用 CASE 表达式, 就可以创建漂亮格式的报表。目标创建报表, 按部门显示各经理手下员工的总工资。如果某个经理在一个特定部门没有下属, 则会返回 0, 否则, 将返回该部门中这个经理所有下属的工资和。另外, 还要在最后添加一列 TOTAL, 它表示报表中所有工资和。满足所有这些需求的解决方案如下:

```
select mgr,
       sum(case deptno when 10 then sal else 0 end) dept10,
       sum(case deptno when 20 then sal else 0 end) dept20,
       sum(case deptno when 30 then sal else 0 end) dept30,
       sum(case flag when '11' then sal else null end) total
  from (
select deptno,mgr,sum(sal) sal,
       cast(grouping(deptno) as char(1))+
       cast(grouping(mgr) as char(1)) flag
  from emp
 where mgr is not null
 group by deptno,mgr with rollup
 ) x
 group by mgr
 order by coalesce(mgr,9999)
```

MGR	DEPT10	DEPT20	DEPT30	TOTAL
7566	0	6000	0	
7698	0	0	6550	
7782	1300	0	0	
7788	0	1100	0	
7839	2450	2975	2850	
7902	0	800	0	
	3750	10875	9400	24025

## 分层查询

本章将介绍如何表示数据的层次关系。当处理分层数据时，检索和显示这些数据比存储它们更难。由于 SQL 不太灵活（SQL 的非递归性质），这种现象更为突出。当处理分层查询时，绝对要利用 RDBMS 提供的方法实现这些操作，否则可能会被为处理分层数据而编写低效的查询和构建令人费解的数据模型搞得焦头烂额。对 PostgreSQL 用户来说，在以后的 PostgreSQL 版本中很可能会添加递归 WITH 子句，这样，就可以采用 DB2 解决方案编写这些查询。

本章将利用每个 RDBMS 提供的函数，介绍如何拆开数据的层次结构。开始讲述之前，先检验表 EMP 及 EMPNO 和 MGR 之间的层次关系：

```
select empno,mgr
  from emp
 order by 2
```

EMPNO	MGR
7788	7566
7902	7566
7499	7698
7521	7698
7900	7698
7844	7698
7654	7698
7934	7782
7876	7788
7566	7839
7782	7839
7698	7839
7369	7902
7839	

如果仔细看，会注意到每个 MGR 值也是一个 EMPNO，这就意味着，表 EMP 中每个员工的经理也是表 EMP 中的一名员工，而且并没有存储在其他位置。MGR 和 EMPNO 之间的关系是父—子的关系，在这种关系中，MGR 是给定 EMPNO 的最直接父母（某些特定员工的经理上面还有经理，而且这些经理也还有经理，依此类推，这样，就构成了一个 n 层的层次关系）。如果某个员工没有经理，则 MGR 为 NULL。

## 13.1 表示父-子关系

### 问题

将父记录的信息跟子记录中的数据放在一起。例如，显示每个员工的姓名及其经理的姓名，返回下列结果集：

```
EMPS_AND_MGRS
-----
FORD works for JONES
SCOTT works for JONES
JAMES works for BLAKE
TURNER works for BLAKE
MARTIN works for BLAKE
WARD works for BLAKE
ALLEN works for BLAKE
MILLER works for CLARK
ADAMS works for SCOTT
CLARK works for KING
BLAKE works for KING
JONES works for KING
SMITH works for FORD
```

### 解决方案

按 MGR 和 EMPNO 自联接 EMP 表，找到每个员工的经理。然后，使用 RDBMS 提供的字符串连接函数，生成结果集中需要的字符串。

#### DB2、Oracle 和 PostgreSQL

自联接 EMP，然后，使用双竖线 (||) 操作符连接字符串：

```
1 select a.ename || ' works for ' || b.ename as emps_and_mgrs
2   from emp a, emp b
3  where a.mgr = b.empno
```

#### MySQL

自联接 EMP，然后，使用函数 CONCAT 连接字符串：

```
1 select concat(a.ename, ' works for ',b.ename) as emps_and_mgrs
2   from emp a, emp b
3  where a.mgr = b.empno
```

#### SQL Server

自联接 EMP 然后，使用加号 (+) 操作符连接字符串：

```
1 select a.ename + ' works for ' + b.ename as emps_and_mgrs
2   from emp a, emp b
3  where a.mgr = b.empno
```

### 讨论

从本质上讲，这些解决方案的实现方式都是相同的。唯一的差别就是字符串连接的方法不同。因此，下面的讨论适用于所有解决方案。



关键是进行 MGR 和 EMPNO 的联接。首先，通过自联接 EMP，建立一个笛卡儿积（下面仅列出了笛卡儿积返回的部分行）：

```
select a.empno, b.empno
  from emp a, emp b
```

EMPNO	MGR
7369	7369
7369	7499
7369	7521
7369	7566
7369	7654
7369	7698
7369	7782
7369	7788
7369	7839
7369	7844
7369	7876
7369	7900
7369	7902
7369	7934
7499	7369
7499	7499
7499	7521
7499	7566
7499	7654
7499	7698
7499	7782
7499	7788
7499	7839
7499	7844
7499	7876
7499	7900
7499	7902
7499	7934

可以看到，笛卡儿积就能返回所有可能的 EMPNO/EMPNO 组合（这样看起来好像表中所有员工都是 EMPNO 7369 的经理，其中也包括 EMPNO 7369 自己）。

下一步筛选结果，只返回每个员工及其经理的 EMPNO 方法是按 MGR 和 EMPNO 做联接：

```
1 select a.empno, b.empno mgr
2   from emp a, emp b
3  where a.mgr = b.empno
```

EMPNO	MGR
7902	7566
7788	7566
7900	7698
7844	7698
7654	7698
7521	7698
7499	7698
7934	7782
7876	7788
7782	7839
7698	7839
7566	7839
7369	7902

现在就有了所有员工及其经理的 EMPNO，只要选出 B.ENAME 而不是 B.EMPNO，就可以返回每个经理的姓名。如果经过一些时间还是没有掌握其中的道理，则可以使用标量子查询（代替自联接）来得到答案：

```
select a.ename,
       (select b.ename
        from emp b
         where b.empno = a.mgr) as mgr
from emp a
```

ENAME	MGR
SMITH	FORD
ALLEN	BLAKE
WARD	BLAKE
JONES	KING
MARTIN	BLAKE
BLAKE	KING
CLARK	KING
SCOTT	JONES
KING	
TURNER	BLAKE
ADAMS	SCOTT
JAMES	BLAKE
FORD	JONES
MILLER	CLARK

标量子查询与自联接基本相同，只是多了一行：该结果集中包含了员工 KING，但自联接方式中却不包含。读者可能要问：“为什么不包含呢？”记住，NULL 不等于任何内容，甚至不等于它自己。在自联接解决方案中，EMPNO 和 MGR 之间使用了等值联接，这样将所有 MGR 为 NULL 的员工筛选掉了。要想在使用自联接方法时也看到员工 KING，必须像下面两个查询一样采用外联接，其中第一个解决方案采用了 ANSI 外联接语法，而第二个方案采用了 Oracle 外联接语法，二者的输出相同，下面给出了第二种查询：

```
/* ANSI */
select a.ename, b.ename mgr
  from emp a left join emp b
    on (a.mgr = b.empno)

/* Oracle */
select a.ename, b.ename mgr
  from emp a, emp b
 where a.mgr = b.empno (+)
```

ENAME	MGR
FORD	JONES
SCOTT	JONES
JAMES	BLAKE
TURNER	BLAKE
MARTIN	BLAKE
WARD	BLAKE
ALLEN	BLAKE
MILLER	CLARK
ADAMS	SCOTT
CLARK	KING
BLAKE	KING
JONES	KING
SMITH	FORD
KING	

## 13.2 表示子—父—祖父关系

### 问题

员工 CLARK 为 KING 工作，本章前面也讲述了他们的关系。假设员工 CLARK 又是另一个员工的经理，怎么办？参阅下列查询：

```
select ename,empno,mgr
  from emp
 where ename in ('KING','CLARK','MILLER')
```

ENAME	EMPNO	MGR
CLARK	7782	7839
KING	7839	
MILLER	7934	7782

可以看到，员工 MILLER 为 CLARK 工作，而 CLARK 又为 KING 工作。现在需要表示从 MILLER 到 KING 的完整层次，返回下列结果集：

```
LEAF__BRANCH_ _ _ROOT
-----
MILLER-->CLARK-->KING
```

然而，单单使用前一节中介绍的自联接方法，不足以显示从顶至底的整个关系。可以编写一个查询，使它包含两个自联接，但现在真正需要的是遍历这种层次关系的通用方法。

### 解决方案

正如标题中所揭示的那样，由于存在 3 层关系，这个问题不同于第一节的问题。如果 RDBMS 没有提供有关遍历树结构数据的功能，则可以采用其他技巧解决这个问题，但一定要添加一个额外的自联接。DB2、SQL Server 和 Oracle 都提供了表示层次的函数。因此，尽管自联接方法在这些 RDBMS 也能用，但不必使用它们。

#### DB2 和 SQL Server

使用递归 WITH 子句，找到 MILLER 的经理 CLARK，然后再找到 CLARK 的经理 KING。在这个解决方案中，使用了 SQL Server 字符串连接操作符“+”：

```
1  with x (tree,mgr,depth)
2  as (
3  select cast(ename as varchar(100)),
4         mgr, 0
5  from emp
6  where ename = 'MILLER'
7  union all
8  select cast(x.tree+'-->'+e.ename as varchar(100)),
9         e.mgr, x.depth+1
10 from emp e, x
11 where x.mgr = e.empno
12 )
13 select tree leaf__branch_ _ _root
14 from x
15 where depth = 2
```

要在 DB2 中实现这种解决方案，唯一需要修改的是使用 DB2 的连接操作符 “||” 除此之外，该解决方案在两个平台上都一样。

## Oracle

使用函数 SYS\_CONNECT\_BY\_PATH, 返回 MILLER、MILLER 的经理 CLARK、CLARK 的经理 KING。使用 CONNECT BY 子句遍历树：

```
1 select ltrim(
2     sys_connect_by_path(ename,'-->'),
3     '-->') leaf__branch_ _ _root
4   from emp
5  where level = 3
6     start with ename = 'MILLER'
7     connect by prior mgr = empno
```

## PostgreSQL 和 MySQL

对表 EMP 进行两次自联接，返回 MILLER、MILLER 的经理 CLARK、CLARK 的经理 KING。下面的解决方案使用了 PostgreSQL 的连接操作符双竖线 (||)：

```
1 select a.ename||'-->'||b.ename
2     ||'-->'||c.ename as leaf__branch_ _ _root
3   from emp a, emp b, emp c
4  where a.ename = 'MILLER'
5     and a.mgr = b.empno
6     and b.mgr = c.empno
```

对于 MySQL 用户，只需使用 CONCAT 函数，该解决方案对 PostgreSQL 也适用。

## 讨论

### DB2 和 SQL Server

这里采用的方法是从叶节点开始一直遍历到根节点（也可以按相反方向遍历）。UNION ALL 的上半部只是找到员工 MILLER (叶节点) 的行；UNION ALL 的下半部查找 MILLER 的经理，然后再查找这个人的经理，这种查找“经理的经理”过程会重复进行，直到最高层经理（根节点）。DEPTH 的值从 0 开始，每找到一个经理会自动递增 1。DEPTH 的值是执行递归查询时 DB2 一直保留下来的。

---

注意：要更深入地了解 WITH 子句的递归用法，可访问 <http://gennick.com/with.htm> 网站，查阅 Jonathan Gennick 的文章“了解 WITH 子句”。

---

接下来，UNION ALL 的第二个查询会把递归视图 X 联接到表 EMP，以便定义父—子关系。这里的查询使用了 SQL Server 的字符串连接操作符，如下所示：

```
with x (tree,mgr,depth)
as (
select cast(ename as varchar(100)),
      mgr, 0
  from emp
 where ename = 'MILLER'
```

```

union all
select cast(x.tree+'-->'||e.ename as varchar(100)),
       e.mgr, x.depth+1
  from emp e, x
 where x.mgr = e.empno
)
select tree leaf____branch_ _ _root
  from x

TREE                DEPTH
-----
MILLER              0
CLARK                1
KING                2

```

至此，问题的实质已经解决：即按从下到上的方向，返回从 MILLER 开始的完整层次关系。下面要做的仅仅是设置格式。由于树遍历是递归进行的，把 EMP 表中当前的 ENAME 与它前一个 ENAME 相联接，会产生下列结果集：

```

with x (tree,mgr,depth)
as (
select cast(ename as varchar(100)),
       mgr, 0
  from emp
 where ename = 'MILLER'
union all
select cast(x.tree+'-->'||e.ename as varchar(100)),
       e.mgr, x.depth+1
  from emp e, x
 where x.mgr = e.empno
)
select depth, tree
  from x

DEPTH TREE
-----
0 MILLER
1 MILLER-->CLARK
2 MILLER-->CLARK-->KING

```

最后一步，保留层次中的最后一行。有很多方式可实现此功能，但该解决方案使用 DEPTH 确定什么时候到达根部（很显然，如果 CLARK 的经理不是 KING 而是其他什么人，那么就可能需要修改对 DEPTH 的筛选条件；关于并不需要这种筛选条件的更通用的解决方案，请参阅下面的内容）。

## Oracle

在 Oracle 解决方案中，CONNECT BY 子句完成了所有功能。从 MILLER 开始，一直遍历到 KING，而无需任何联接。CONNECT BY 子句中的表达式定义了数据的关系，也指明了如何遍历树：

```

select ename
  from emp
 start with ename = 'MILLER'
 connect by prior mgr = empno

```

```
ENAME
-----
MILLER
CLARK
KING
```

关键字PRIOR用于访问上一层的记录,这样,对于给定的EMPNO,可以使用PRIOR MGR访问该员工的经理编号。在这个例子中,CONNECT BY PRIOR MGR = EMPNO子句用于表示父与子之间的联接。

注意: 有关更多的 CONNECT BY 及其相关特征, 请参阅下列 Oracle 技术网络文章: [http://www.oracle.com/technology/oramag/webcolumns/2003/techarticles/gennick\\_connectby.html](http://www.oracle.com/technology/oramag/webcolumns/2003/techarticles/gennick_connectby.html) 网站的“Querying Hierarchies: Top-of-the-Line Support”以及 [http://www.oracle.com/technology/oramag/webcolumns/2003/techarticles/gennick\\_connectby\\_10g.html](http://www.oracle.com/technology/oramag/webcolumns/2003/techarticles/gennick_connectby_10g.html) 网站的“New CONNECT BY Features in Oracle Database 10g”。

至此,已经成功地揭示了从 MILLER 至 KING 的完整层次,问题的大部分已经解决。剩下的就是设置格式。使用函数SYS\_CONNECT\_BY\_PATH,把每个ENAME附加到它的前一个ENAME上:

```
select sys_connect_by_path(ename,'-->') tree
       from emp
       start with ename = 'MILLER'
       connect by prior mgr = empno

TREE
-----
-->MILLER
-->MILLER-->CLARK
-->MILLER-->CLARK-->KING
```

由于只对完整的层次感兴趣,因此可以按伪列LEVEL进行筛选(下一个问题将给出更通用的方案):

```
select sys_connect_by_path(ename,'-->') tree
       from emp
       where level = 3
       start with ename = 'MILLER'
       connect by prior mgr = empno

TREE
-----
-->MILLER-->CLARK-->KING
```

最后一步,使用LTRIM函数,从结果集中删除前导符“-->”。

## PostgreSQL 和 MySQL

对于分层查询,如果没有内置函数支持,那么必须进行n次自联接,以便返回整个树(n是叶子和根之间的节点数,其中包括根本身;在这个例子中,叶子是MILLER,CLARK是一个分支节点,KING是根节点,所以距离是两个节点,即n=2)。这个解决方案采用了上一节中的技巧,只是多了一个自联接:

```
select a.ename as leaf,
       b.ename as branch,
       c.ename as root
from emp a, emp b, emp c
where a.ename = 'MILLER'
      and a.mgr = b.empno
      and b.mgr = c.empno
```

LEAF	BRANCH	ROOT
MILLER	CLARK	KING

下一步，也就是最后一步，对于 PostgreSQL，使用连接操作符“||”设置输出格式；对于 MySQL，使用 CONCAT 函数设置输出的格式。这种查询的缺点是：如果更改了层次，例如，在 CLARK 和 KING 之间还有另一个节点，则需要再加一个联接，以便返回整个树。因此，对于层次查询，最好使用内置函数。

## 13.3 创建表的分层视图

### 问题

返回一个结果集，它描述整个表的层次。在 EMP 表中，员工 KING 没有经理，所以 KING 是根节点。从 KING 开始，显示 KING 下面的所有员工以及 KING 下属的所有员工（如果有的话）。最后，返回下列结果集：

```
EMP_TREE
-----
KING
KING - BLAKE
KING - BLAKE - ALLEN
KING - BLAKE - JAMES
KING - BLAKE - MARTIN
KING - BLAKE - TURNER
KING - BLAKE - WARD
KING - CLARK
KING - CLARK - MILLER
KING - JONES
KING - JONES - FORD
KING - JONES - FORD - SMITH
KING - JONES - SCOTT
KING - JONES - SCOTT - ADAMS
```

### 解决方案

#### DB2 和 SQL Server

使用递归 WITH 子句，建立以 KING 开头的层次，然后显示所有员工。下面的解决方案使用 DB2 连接操作符“||”。而 SQL Server 用户需要使用连接操作符“+”，除此之外对于这两个 RDBMS，该解决方案都一样：

```
1  with x (ename,empno)
2    as (
3  select cast(ename as varchar(100)),empno
4    from emp
5   where mgr is null
6   union all
```

```

7  select cast(x.ename||' - '||e.ename as varchar(100)),
8         e.empno
9         from emp e, x
10        where e.mgr = x.empno
11      )
12  select ename as emp_tree
13        from x
14      order by 1

```

## Oracle

使用 CONNECT BY 函数定义层次。使用 SYS\_CONNECT\_BY\_PATH 函数设置输出格式：

```

1  select ltrim(
2         sys_connect_by_path(ename,' - '),
3         ' - ') emp_tree
4        from emp
5      start with mgr is null
6     connect by prior empno=mgr
7     order by 1

```

这个解决方案不同于上一节的方案，它不包含按伪列 LEVEL 的筛选，这样就会显示所有的树（PRIOR EMPNO=MGR）。

## PostgreSQL

使用三个 UNION 和多个自联接：

```

1  select emp_tree
2        from (
3  select ename as emp_tree
4        from emp
5       where mgr is null
6  union
7  select a.ename||' - '||b.ename
8        from emp a
9        join
10       emp b on (a.empno=b.mgr)
11       where a.mgr is null
12  union
13  select rtrim(a.ename||' - '||b.ename
14             ||' - '||c.ename,' - ')
15        from emp a
16        join
17       emp b on (a.empno=b.mgr)
18        left join
19       emp c on (b.empno=c.mgr)
20       where a.ename = 'KING'
21  union
22  select rtrim(a.ename||' - '||b.ename||' - '||
23             c.ename||' - '||d.ename,' - ')
24        from emp a
25        join
26       emp b on (a.empno=b.mgr)
27        join
28       emp c on (b.empno=c.mgr)
29        left join
30       emp d on (c.empno=d.mgr)
31       where a.ename = 'KING'
32      ) x
33  where tree is not null
34  order by 1

```



## MySQL

使用三个 UNION 和多个自联接：

```

1  select emp_tree
2  from (
3  select ename as emp_tree
4  from emp
5  where mgr is null
6  union
7  select concat(a.ename, ' - ', b.ename)
8  from emp a
9  join
10     emp b on (a.empno=b.mgr)
11  where a.mgr is null
12  union
13  select concat(a.ename, ' - ',
14              b.ename, ' - ', c.ename)
15  from emp a
16  join
17     emp b on (a.empno=b.mgr)
18  left join
19     emp c on (b.empno=c.mgr)
20  where a.ename = 'KING'
21  union
22  select concat(a.ename, ' - ', b.ename, ' - ',
23              c.ename, ' - ', d.ename)
24  from emp a
25  join
26     emp b on (a.empno=b.mgr)
27  join
28     emp c on (b.empno=c.mgr)
29  left join
30     emp d on (c.empno=d.mgr)
31  where a.ename = 'KING'
32  ) x
33  where tree is not null
34  order by 1

```

## 讨论

### DB2 和 SQL Server

在递归视图 X 中，UNION ALL 的上半部用于识别根行（员工 KING）。下一步，通过把递归视图 X 联接到表 EMP，从而找到 KING 的下属及其下属的下属（如果有的话）。递归一直执行下去，直到返回所有员工。如果不进行格式设置，那么递归视图 X 返回的最终结果集如下：

```

with x (ename,empno)
as (
select cast(ename as varchar(100)),empno
from emp
where mgr is null
union all
select cast(e.ename as varchar(100)),e.empno
from emp e, x
where e.mgr = x.empno
)
select ename emp_tree
from x

```

```

EMP_TREE
-----
KING
JONES
SCOTT
ADAMS
FORD
SMITH
BLAKE
ALLEN
WARD
MARTIN
TURNER
JAMES
CLARK
MILLER

```

现在已经返回了层次中的所有行（可能会很有用），但如果不进行格式设置，就不知道其中哪些是经理。如果把每个员工与他的经理连接起来，得到的输出更有意义。在递归视图 X 中，UNION ALL 的下半部使用了 SELECT 子句，其中包含 cast(x.ename+', '+e.ename as varchar(100))，它会产生想要的输出结果。

要解决这种类型的问题，WITH 子句非常有用，因为即使层次变了（例如，叶节点变为分支节点），也无需修改查询。

## Oracle

CONNECT BY 子句将返回层次中的行。START WITH 子句定义了根行。如果解决方案没有使用 SYS\_CONNECT\_BY\_PATH，则会返回正确的行（可能会很有用），但没有设置格式，以表示行之间的关系：

```

select ename emp_tree
  from emp
 start with mgr is null
connect by prior empno = mgr

EMP_TREE
-----
KING
JONES
SCOTT
ADAMS
FORD
SMITH
BLAKE
ALLEN
WARD
MARTIN
TURNER
JAMES
CLARK
MILLER

```

如果使用伪列 LEVEL 和函数 LPAD，就能够更清晰地查看层次，最终也会明白 SYS\_CONNECT\_BY\_PATH 能够返回上述结果集的原因：

```

select lpad(' ', 2*level, ' ')||ename emp_tree
  from emp
 start with mgr is null
connect by prior empno = mgr

```

```
EMP_TREE
-----
..KING
....JONES
.....SCOTT
.....ADAMS
.....FORD
.....SMITH
....BLAKE
.....ALLEN
.....WARD
.....MARTIN
.....TURNER
.....JAMES
....CLARK
.....MILLER
```

该输出结果用缩排表明了经理是谁，缩排的方式是把下属嵌套于其上级之下，例如，KING不为任何人工作，JONES为KING工作，SCOTT为JONES工作，ADAMS为SCOTT工作。

如果在该解决方案中使用了SYS\_CONNECT\_BY\_PATH，当查看输出时，会看到SYS\_CONNECT\_BY\_PATH会卷起层次。每到一个新节点时，也会看到它前面的所有节点：

```
KING
KING - JONES
KING - JONES - SCOTT
KING - JONES - SCOTT - ADAMS
```

**注意：**在 Oracle8i Database 或较早版本中，可以采用 PostgreSQL 解决方案解决该问题。另外，由于在 Oracle 的较早版本中可以使用 CONNECT BY，则可以使用 LEVEL 和 RPAD/LPAD 设置格式（尽管重新生成由 SYS\_CONNECT\_BY\_PATH 创建的输出还需要一点儿额外工作）。

## PostgreSQL 和 MySQL

除了 SELECT 子句中的字符串连接，PostgreSQL 和 MySQL 的解决方案相同。首先，为所有分支确定最大节点数。在编写查询之前，必须手动做这件事。如果查看 EMP 表中的数据，会看到员工 ADAM 和 SMITH 都是位于最深层的叶子节点（查阅 Oracle 讨论内容，会看到 EMP 层次的优美格式树）。如果查看员工 ADAMS，会看到 ADAMS 为 SCOTT 工作，SCOTT 为 JONES 工作，JONES 为 KING 工作，因此深度为 4。要表示一个 4 层深度的层次，必须对表 EMP 的 4 个实例进行自联接，而且必须编写 4 部分 UNION 查询。4 路自联接的结果（由上至下，最后一个 UNION 以下部分）如下所示（对于 MySQL 用户，可采用 PostgreSQL 语法，用“||”代替 CONCAT 函数调用）：

```
select rtrim(a.ename||' - '||b.ename||' - '||
        c.ename||' - '||d.ename,' - ') as max_depth_4
from emp a
join
emp b on (a.empno=b.mgr)
join
emp c on (b.empno=c.mgr)
left join
```

```

emp d on (c.empno=d.mgr)
where a.ename = 'KING'
MAX_DEPTH_4
-----
KING - JONES - FORD - SMITH
KING - JONES - SCOTT - ADAMS
KING - BLAKE - TURNER
KING - BLAKE - ALLEN
KING - BLAKE - WARD
KING - CLARK - MILLER
KING - BLAKE - MARTIN
KING - BLAKE - JAMES

```

按 A.ENAME 进行的筛选是必要的，这样是为了确保根节点行是 KING，而不是其他员工。如果查看上面的结果集，并把它与最终结果集相比较，会发现，丢掉了一些三层深的层次：KING - JONES - FORD 和 KING - JONES - SCOTT。要在最终结果集中引入这些行，必须编写一个与上述查询相似的查询，但少一个联接（仅自联接表 EMP 的 3 个实例，而不是 4 个）。该查询的结果集如下所示：

```

select rtrim(a.ename||' - '||b.ename
           ||' - '||c.ename,' - ') as max_depth_3
  from emp a
      join
      emp b on (a.empno=b.mgr)
      left join
      emp c on (b.empno=c.mgr)
 where a.ename = 'KING'
MAX_DEPTH_3
-----
KING - BLAKE - ALLEN
KING - BLAKE - WARD
KING - BLAKE - MARTIN
KING - JONES - SCOTT
KING - BLAKE - TURNER
KING - BLAKE - JAMES
KING - JONES - FORD
KING - CLARK - MILLER

```

与前面的查询一样，按 A.ENAME 的筛选也是必要的，以确保根行节点是 KING。注意，在上面的查询和 4 路 EMP 联接之间存在一些重叠行。要去掉这些多余行，只需使用 UNION 把两个查询的结果合并起来：

```

select rtrim(a.ename||' - '||b.ename
           ||' - '||c.ename,' - ') as partial_tree
  from emp a
      join
      emp b on (a.empno=b.mgr)
      left join
      emp c on (b.empno=c.mgr)
 where a.ename = 'KING'
union
select rtrim(a.ename||' - '||b.ename||' - '||
           c.ename||' - '||d.ename,' - ')
  from emp a
      join
      emp b on (a.empno=b.mgr)
      join
      emp c on (b.empno=c.mgr)
      left join
      emp d on (c.empno=d.mgr)
 where a.ename = 'KING'

```

```
PARTIAL_TREE
-----
KING - BLAKE - ALLEN
KING - BLAKE - JAMES
KING - BLAKE - MARTIN
KING - BLAKE - TURNER
KING - BLAKE - WARD
KING - CLARK - MILLER
KING - JONES - FORD
KING - JONES - FORD - SMITH
KING - JONES - SCOTT
KING - JONES - SCOTT - ADAMS
```

至此，树几乎遍历完了。下一步，用 KING 作为根节点，返回那些表示两层深度的层次（即直接为 KING 工作的员工）。用以返回这些行的查询如下：

```
select a.ename||' - '||b.ename as max_depth_2
  from emp a
    join
      emp b on (a.empno=b.mgr)
 where a.mgr is null
MAX_DEPTH_2
-----
KING - JONES
KING - BLAKE
KING - CLARK
```

下一步，使用 UNION，把上面的查询与 PARTIAL\_TREE 的结果合并起来：

```
select a.ename||' - '||b.ename as partial_tree
  from emp a
    join
      emp b on (a.empno=b.mgr)
 where a.mgr is null
union
select rtrim(a.ename||' - '||b.ename
            ||' - '||c.ename,' - ')
  from emp a
    join
      emp b on (a.empno=b.mgr)
    left join
      emp c on (b.empno=c.mgr)
 where a.ename = 'KING'
union
select rtrim(a.ename||' - '||b.ename||' - '||
            c.ename||' - '||d.ename,' - ')
  from emp a
    join
      emp b on (a.empno=b.mgr)
    join
      emp c on (b.empno=c.mgr)
    left join
      emp d on (c.empno=d.mgr)
 where a.ename = 'KING'
PARTIAL_TREE
-----
KING - BLAKE
KING - BLAKE - ALLEN
KING - BLAKE - JAMES
KING - BLAKE - MARTIN
KING - BLAKE - TURNER
KING - BLAKE - WARD
KING - CLARK
KING - CLARK - MILLER
KING - JONES
```

```
KING - JONES - FORD
KING - JONES - FORD - SMITH
KING - JONES - SCOTT
KING - JONES - SCOTT - ADAMS
```

最后一步，使用 UNION KING 联到 PARTIAL\_TREE 的顶部，以返回想要的结果集。

## 13.4 为给定父行找到所有子行

### 问题

找到直接及间接（即 JONES 下属的下属）为 JONES 工作的所有员工。JONES 下属的员工列表如下所示（结果集中也包含 JONES）：

```
ENAME
-----
JONES
SCOTT
ADAMS
FORD
SMITH
```

### 解决方案

能够移到树的绝对顶部和底部是非常有用的。对于这个解决方案，不需要特殊的格式设置。目标只是返回位于员工 JONES 下属的所有员工，其中包括 JONES 自己。这种类型的查询展示了递归 SQL 扩展的有用性，如 Oracle 的 CONNECT BY 和 SQL Server/DB2 的 WITH 子句等。

#### DB2 和 SQL Server

使用递归 WITH 子句，找到 JONES 下属的所有员工。在两个联合查询的第一个查询中，用 WHERE ENAME = 'JONES' 指定从 JONES 开始：

```
1  with x (ename,empno)
2  as (
3  select ename,empno
4  from emp
5  where ename = 'JONES'
6  union all
7  select e.ename, e.empno
8  from emp e, x
9  where x.empno = e.mgr
10 )
11 select ename
12 from x
```

#### Oracle

使用 CONNECT BY 子句，指定 START WITH ENAME = 'JONES'，从而找到 JONES 下属的所有员工：

```
1  select ename
2  from emp
3  start with ename = 'JONES'
4  connect by prior empno = mgr
```

## PostgreSQL 和 MySQL

必须预先知道树中有多少个节点。下面的查询展示了如何确定层次的深度：

```

/* find JONES' EMPNO */
select ename,empno,mgr
  from emp
 where ename = 'JONES'

ENAME          EMPNO          MGR
-----
JONES          7566          7839

/* are there any employees who work directly under JONES? */
select count(*)
  from emp
 where mgr = 7566

COUNT(*)
-----
2

/* there are two employees under JONES, find their EMPNOs */
select ename,empno,mgr
  from emp
 where mgr = 7566

ENAME          EMPNO          MGR
-----
SCOTT          7788          7566
FORD           7902          7566

/* are there any employees under SCOTT or FORD? */
select count(*)
  from emp
 where mgr in (7788,7902)

COUNT(*)
-----
2

/* there are two employees under SCOTT or FORD, find their EMPNOs */
select ename,empno,mgr
  from emp
 where mgr in (7788,7902)

ENAME          EMPNO          MGR
-----
SMITH          7369          7902
ADAMS          7876          7788

/* are there any employees under SMITH or ADAMS? */
select count(*)
  from emp
 where mgr in (7369,7876)

COUNT(*)
-----
0

```

该层次从 JONES 开始，至 SMITH 和 ADAMS 结束，这样该层次深度就为 3。有了这个深度之后，就可以开始从上至下遍历该层次了。

首先，对表 EMP 进行两次自联接。然后，反转置内联视图 X，以便把两行三列转换为六行一列（在 PostgreSQL 中，可以不查询 T100 基干表，而使用 GENERATE\_SERIES(1,6)）：

```

1 select distinct
2     case t100.id
3         when 1 then root
4         when 2 then branch
5         else     leaf
6     end as JONES_SUBORDINATES
7   from (
8   select a.ename as root,
9         b.ename as branch,
10        c.ename as leaf
11   from emp a, emp b, emp c
12  where a.ename = 'JONES'
13        and a.empno = b.mgr
14        and b.empno = c.mgr
15        ) x,
16        t100
17  where t100.id <= 6

```

另外，也可以使用视图，并使用 UNION 合并结果集。如果创建了下列视图：

```

create view v1
as
select ename,mgr,empno
  from emp
 where ename = 'JONES'
create view v2
as
select ename,mgr,empno
  from emp
 where mgr = (select empno from v1)

create view v3
as
select ename,mgr,empno
  from emp
 where mgr in (select empno from v2)

```

那么解决方案就会变为：

```

select ename from v1
union
select ename from v2
union
select ename from v3

```

## 讨论

### DB2 和 SQL Server

递归 WITH 子句会使这个问题更容易解决。WITH 子句的第一部分，也即 UNION ALL 的上半部，返回员工 JONES 的行。为查看员工姓名故需要返回 ENAME，也要返回 EMPNO 用以进行联接。UNION ALL 的下半部把 EMP.MGR 递归联接到 X.EMPNO，联接条件将持续到结果集处理完毕为止。

### Oracle

START WITH 子句会通知查询，JONES 是根节点。CONNECT BY 子句中的条件将驱动遍历树，而且一直运行，直到不再满足条件。



PostgreSQL 和 MySQL

这里采用的技巧与本章 13.2 节中的技巧相同。主要缺点是必须预先知道层次的深度。

13.5 确定哪些行是叶节点、分支节点及根节点

问题

确定给定行属于哪种类型的节点：叶节点、分支节点及根节点。对于这个例子，叶节点表示该员工不是经理；分支节点上的员工既是经理，又有经理根节点是没有经理的员工。通过返回 1 (TRUE) 或 0 (FALSE)，来表示层次中每行的状态。应返回下列结果集：

ENAME	IS_LEAF	IS_BRANCH	IS_ROOT
KING	0	0	1
JONES	0	1	0
SCOTT	0	1	0
FORD	0	1	0
CLARK	0	1	0
BLAKE	0	1	0
ADAMS	1	0	0
MILLER	1	0	0
JAMES	1	0	0
TURNER	1	0	0
ALLEN	1	0	0
WARD	1	0	0
MARTIN	1	0	0
SMITH	1	0	0

解决方案

请注意，EMP表是树状层次模型，而不是递归层次模型，根节点的MGR值为NULL。如果EMP是递归层次模型的话，根节点应有自引用（即员工KING的MGR值将是KING的EMPNO）。实际上，自引用并不直观，故这里将根节点的MGR设为NULL值。对于在Oracle中使用CONNECT BY以及在DB2/SQL Server中使用WITH的情况，树状层次比递归层次更容易处理，而且也更有效。对于在递归层次模型中使用CONNECT BY或WITH的情况，请务必当心：在SQL可能会死循环；如果一定要采用递归层次，代码中必需考虑避免这样的循环。

DB2、PostgreSQL、MySQL 和 SQL Server

使用三个标量子查询，确定正确的“布尔”值（1或0），以返回每个节点类型：

```
1  select e.ename,
2      (select sign(count(*)) from emp d
3         where 0 =
4             (select count(*) from emp f
5                where f.mgr = e.empno)) as is_leaf,
6      (select sign(count(*)) from emp d
7         where d.mgr = e.empno
8           and e.mgr is not null) as is_branch,
9      (select sign(count(*)) from emp d
10         where d.empno = e.empno
11           and d.mgr is null) as is_root
12  from emp e
```

```
13 order by 4 desc,3 desc
```

## Oracle

标量子查询解决方案对 Oracle 也同样适用，如果使用的是 Oracle Database 10g 之前的版本，只能采用这种方案。下面的解决方案将重点介绍 Oracle 提供的内置函数（Oracle Database 10g 中引入的），以识别根行和叶行。这两个函数分别为 CONNECT\_BY\_ROOT 和 CONNECT\_BY\_ISLEAF：

```
1 select ename,
2        connect_by_isleaf is_leaf,
3        (select count(*) from emp e
4         where e.mgr = emp.empno
5         and emp.mgr is not null
6         and rownum = 1) is_branch,
7        decode(ename,connect_by_root(ename),1,0) is_root
8      from emp
9      start with mgr is null
10     connect by prior empno = mgr
11     order by 4 desc, 3 desc
```

## 讨论

### DB2、PostgreSQL、MySQL 和 SQL Server

这个解决方案只是采用“问题”部分定义的规则，确定叶节点、分支节点和根节点。第一步，确定员工是否是叶节点。如果员工不是经理（没有人在他手下工作），那么他就是叶节点。第一个标量子查询 IS\_LEAF 如下所示：

```
select e.ename,
       (select sign(count(*)) from emp d
        where 0 =
          (select count(*) from emp f
           where f.mgr = e.empno)) as is_leaf
  from emp e
 order by 2 desc
```

ENAME	IS_LEAF
SMITH	1
ALLEN	1
WARD	1
ADAMS	1
TURNER	1
MARTIN	1
JAMES	1
MILLER	1
JONES	0
BLAKE	0
CLARK	0
FORD	0
SCOTT	0
KING	0

由于 IS\_LEAF 的输出应该是 0 或 1，用 SIGN 获取 COUNT(\*) 的正负号是必要的，否则，对于叶节点，会得到 14，而不是 1。还有另一种方法，由于只想返回 0 或 1，因此还可以使用仅包含一行的表作为计数对象。例如：

```
select e.ename,
       (select count(*) from t1 d
        where not exists
          (select null from emp f
           where f.mgr = e.empno)) as is_leaf
  from emp e
 order by 2 desc
```

ENAME	IS_LEAF
SMITH	1
ALLEN	1
WARD	1
ADAMS	1
TURNER	1
MARTIN	1
JAMES	1
MILLER	1
JONES	0
BLAKE	0
CLARK	0
FORD	0
SCOTT	0
KING	0

下一步，找到分支节点。如果员工是经理（有人为他工作），他们也为其他人工作，那么该员工就是分支节点。标量子查询 IS\_BRANCH 的结果如下所示：

```
select e.ename,
       (select sign(count(*)) from emp d
        where d.mgr = e.empno
          and e.mgr is not null) as is_branch
  from emp e
 order by 2 desc
```

ENAME	IS_BRANCH
JONES	1
BLAKE	1
SCOTT	1
CLARK	1
FORD	1
SMITH	0
TURNER	0
MILLER	0
JAMES	0
ADAMS	0
KING	0
ALLEN	0
MARTIN	0
WARD	0

同样，用 SIGN 获取 COUNT(\*) 的正负号是必要的，否则，当节点是分支时，会得到（有可能）大于 1 的值。与标量子查询 IS\_LEAF 一样，也可以使用仅包含一行的表，以避免使用 SIGN。下面的解决方案使用了名为 dual 的单行表：

```
select e.ename,
       (select count(*) from t1 t
        where exists (
          select null from emp f
          where f.mgr = e.empno
            and e.mgr is not null)) as is_branch
  from emp e
 order by 2 desc
```

ENAME	IS_BRANCH
-----	-----
JONES	1
BLAKE	1
SCOTT	1
CLARK	1
FORD	1
SMITH	0
TURNER	0
MILLER	0
JAMES	0
ADAMS	0
KING	0
ALLEN	0
MARTIN	0
WARD	0

最后一步，找到根节点。把根节点定义为这样的员工：他是经理，而且不为其他任何人工作。在表 EMP 中，只有 KING 是根节点。标量子查询 IS\_ROOT 如下所示：

```
select e.ename,
       (select sign(count(*)) from emp d
        where d.empno = e.empno
              and d.mgr is null) as is_root
  from emp e
 order by 2 desc
```

ENAME	IS_ROOT
-----	-----
KING	1
SMITH	0
ALLEN	0
WARD	0
JONES	0
TURNER	0
JAMES	0
MILLER	0
FORD	0
ADAMS	0
MARTIN	0
BLAKE	0
CLARK	0
SCOTT	0

由于 EMP 是一个 14 行的小表，所以能够很容易看到员工 KING 是唯一的根节点，因此，在这个例子中，不需要获取 COUNT(\*) 符号的 SIGN。如果有多个根节点，那么就需要使用 SIGN，也可以像前面介绍的 IS\_BRANCH 和 IS\_LEAF 一样在标量子查询中使用单行表。

## Oracle

在 Oracle Database 10g 之前的版本中，可以参阅其他 RDBMS 的讨论，该解决方案在 Oracle 中也同样适用（不必进行修改）。对于 Oracle Database 10g 及更高版本，可以使用两个函数，它们会使识别根节点和叶节点更为容易，这两个函数分别为：CONNECT\_BY\_ROOT 和 CONNECT\_BY\_ISLEAF。到本书编写时，在 SQL 语句中必须使用 CONNECT BY，这样才能够使用 CONNECT\_BY\_ROOT 和 CONNECT\_BY\_ISLEAF。首先，使用 CONNECT\_BY\_ISLEAF 找到叶节点，如下：

```
select ename,
       connect_by_isleaf is_leaf
  from emp
 start with mgr is null
 connect by prior empno = mgr
 order by 2 desc
```

ENAME	IS_LEAF
ADAMS	1
SMITH	1
ALLEN	1
TURNER	1
MARTIN	1
WARD	1
JAMES	1
MILLER	1
KING	0
JONES	0
BLAKE	0
CLARK	0
FORD	0
SCOTT	0

下一步，使用标量子查询，找到分支节点。属于分支节点的员工是经理，但他也为其他人工作：

```
select ename,
       (select count(*) from emp e
        where e.mgr = emp.empno
          and emp.mgr is not null
          and rownum = 1) is_branch
  from emp
 start with mgr is null
 connect by prior empno = mgr
 order by 2 desc
```

ENAME	IS_BRANCH
JONES	1
SCOTT	1
BLAKE	1
FORD	1
CLARK	1
KING	0
MARTIN	0
MILLER	0
JAMES	0
TURNER	0
WARD	0
ADAMS	0
ALLEN	0
SMITH	0

必须按 ROWNUM 进行筛选，以确保返回 0 或 1，而不是其他值。

最后一步，使用函数 CONNECT\_BY\_ROOT 识别根节点。该解决方案查找根节点的 ENAME，而且把它与查询返回的所有行相比较。如果匹配，则行是根节点：

```
select ename,
       decode(ename,connect_by_root(ename),1,0) is_root
  from emp
 start with mgr is null
 connect by prior empno = mgr
```

**order by 2 desc**

ENAME	IS_ROOT
KING	1
JONES	0
SCOTT	0
ADAMS	0
FORD	0
SMITH	0
BLAKE	0
ALLEN	0
WARD	0
MARTIN	0
TURNER	0
JAMES	0
CLARK	0
MILLER	0

在 Oracle9i Database 或更高版本中，可以用 SYS\_CONNECT\_BY\_PATH 函数代替 CONNECT\_BY\_ROOT。上述方案的 Oracle9i Database 版本如下：

```
select ename,
       decode(substr(root,1,instr(root,',')-1),NULL,1,0) root
  from (
select ename,
       ltrim(sys_connect_by_path(ename,','),'') root
  from emp
 start with mgr is null
 connect by prior empno=mgr
 )
```

ENAME	ROOT
KING	1
JONES	0
SCOTT	0
ADAMS	0
FORD	0
SMITH	0
BLAKE	0
ALLEN	0
WARD	0
MARTIN	0
TURNER	0
JAMES	0
CLARK	0
MILLER	0

SYS\_CONNECT\_BY\_PATH 函数能够从根值开始对层进行上卷，如下所示：

```
select ename,
       ltrim(sys_connect_by_path(ename,','),'') path
  from emp
 start with mgr is null
 connect by prior empno=mgr
```

ENAME	PATH
KING	KING
JONES	KING,JONES
SCOTT	KING,JONES,SCOTT
ADAMS	KING,JONES,SCOTT,ADAMS
FORD	KING,JONES,FORD
SMITH	KING,JONES,FORD,SMITH
BLAKE	KING,BLAKE

ALLEN	KING, BLAKE, ALLEN
WARD	KING, BLAKE, WARD
MARTIN	KING, BLAKE, MARTIN
TURNER	KING, BLAKE, TURNER
JAMES	KING, BLAKE, JAMES
CLARK	KING, CLARK
MILLER	KING, CLARK, MILLER

要得到根行，只要从 PATH 值中提取第一个 ENAME 的子串：

```
select ename,
       substr(root,1,instr(root,',')-1) root
  from (
select ename,
       ltrim(sys_connect_by_path(ename,','),'') root
  from emp
 start with mgr is null
 connect by prior empno=mgr
       )
```

ENAME	ROOT
KING	
JONES	KING
SCOTT	KING
ADAMS	KING
FORD	KING
SMITH	KING
BLAKE	KING
ALLEN	KING
WARD	KING
MARTIN	KING
TURNER	KING
JAMES	KING
CLARK	KING
MILLER	KING

最后一步，如果 ROOT 列为 NULL，则将该行标记为根行。

## 第 14 章

# 若干另类目标

本章要介绍的查询不宜放到其他章节中，一方面它们可以归入的章节已经很长，另一方面，它们解决的问题比实际问题更有趣。本章将是“有趣的”一章，这里涉及的问题可能会在现实中用到，也可能用不到；然而，这里还是要引入这些有趣的查询。

## 14.1 使用 SQL Server 的 PIVOT 运算符创建交叉表报表

### 问题

创建一个交叉表报表，把结果集的行转换为列。虽然前面已经介绍过传统的转置方法，但还是想尝试以下其他方法。具体地说，不使用 CASE 表达式，也不使用联接，要返回下列结果集：

DEPT_10	DEPT_20	DEPT_30	DEPT_40
3	5	6	0

### 解决方案

在不使用 CASE 表达式、也不使用额外联接的情况下，使用 PIVOT 运算符就可以创建想要的结果集：

```

1 select [10] as dept_10,
2        [20] as dept_20,
3        [30] as dept_30,
4        [40] as dept_40
5   from (select deptno, empno from emp) driver
6  pivot (
7    count(driver.empno)
8    for driver.deptno in ( [10],[20],[30],[40] )
9  ) as empPivot

```



讨论

读者可能对PIVOT运算符感到很陌生，但从技术上讲，它在该解决方案中实现的功能与下面列出的变换查询相同：

```
select sum(case deptno when 10 then 1 else 0 end) as dept_10,
       sum(case deptno when 20 then 1 else 0 end) as dept_20,
       sum(case deptno when 30 then 1 else 0 end) as dept_30,
       sum(case deptno when 40 then 1 else 0 end) as dept_40
from emp
```

DEPT_10	DEPT_20	DEPT_30	DEPT_40
3	5	6	0

至此，已经明白了实际发生的事情，下面就分步说明PIVOT运算符的功能。该解决方案的第5行显示了内联视图DRIVER：

```
from (select deptno, empno from emp) driver
```

这里，选择了“driver”别名，主要因为这个内联视图（或表表达式）中的行都直接送到了PIVOT运算符。通过计算第8行FOR列表中的项，PIVOT运算符会把行转换为列（如下所示）：

```
for driver.deptno in ( [10],[20],[30],[40] )
```

计算将如下进行：

- 1. 对于DEPTNO值为10的行，执行聚集运算（COUNT(DRIVER.EMPNO)）。
- 2. 对DEPTNO 20、30和40重复上一步操作。

第8行括号中列出的项，不仅定义要进行聚集操作的值，而且这些项也会成为结果集中的列名（不带方括号）。在这个解决方案中，SELECT子句引用了FOR列表中的项，而且为它起了别名。如果没有给FOR列表中的项起别名，则这些列名就成为FOR列表中的项。

有趣的是，由于内联视图DRIVER就是一个内联视图，所以可以使用更复杂的SQL。例如，修改结果集，让实际的部门名成为列名。下面列出了表DEPT中的行：

```
select * from dept
```

DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON

使用PIVOT，返回下列结果集：

ACCOUNTING	RESEARCH	SALES	OPERATIONS
3	5	6	0

由于内联视图 DRIVER 能够成为任意有效的表表达式，因此可以进行表 EMP 到表 DEPT 的联接，然后让 PIVOT 处理这些行。下列查询将返回想要的结果集：

```
select [ACCOUNTING] as ACCOUNTING,
       [SALES]       as SALES,
       [RESEARCH]    as RESEARCH,
       [OPERATIONS]  as OPERATIONS
  from (
        select d.dname, e.empno
          from emp e,dept d
         where e.deptno=d.deptno
       ) driver
 pivot (
   count(driver.empno)
 for driver.dname in ([ACCOUNTING],[SALES],[RESEARCH],[OPERATIONS])
 ) as empPivot
```

可以看到，PIVOT 转置结果集时进行了有趣的旋转。不管是喜欢这种方法，还是喜欢传统转置方法，在工具箱中多一种工具还是蛮不错的。

## 14.2 使用 SQL Server 的 UNPIVOT 运算符反转置交叉表报表

### 问题

有一个转置结果集（又称胖表），希望反转置该结果集。例如，对于包含一行四列的结果集，返回一个四行两列的结果集。对上一节的结果集进行如下转换：

ACCOUNTING	RESEARCH	SALES	OPERATIONS
3	5	6	0

为：

DNAME	CNT
ACCOUNTING	3
RESEARCH	5
SALES	6
OPERATIONS	0

### 解决方案

SQL Server 既然提供 PIVOT 运算符，也应该会提供 UNPIVOT 运算符，不是吗？要反转置结果集，只需把它当作驱动程序表，让 UNPIVOT 运算符完成所有工作。现在要做的是指定列名：

```
1 select DNAME, CNT
2   from (
3     select [ACCOUNTING] as ACCOUNTING,
4            [SALES]       as SALES,
5            [RESEARCH]    as RESEARCH,
6            [OPERATIONS]  as OPERATIONS
7     from (
8           select d.dname, e.empno
9             from emp e,dept d
```

```

10             where e.deptno=d.deptno
11
12         ) driver
13     pivot (
14         count(driver.empno)
15         for driver.dname in
16         ([ACCOUNTING],[SALES],[RESEARCH],[OPERATIONS])
17         ) as empPivot
18 ) new_driver
19 unpivot (cnt for dname in (ACCOUNTING,SALES,RESEARCH,OPERATIONS)
20 ) as un_pivot

```

在本节之前，以上内容已经出现过，因为内联视图 NEW\_DRIVER 就是上一节给出的代码（如果不能理解，请查阅上一节）。第 3~16 行的代码都曾介绍过，这里唯一的新语法位于第 18 行，即 UNPIVOT。

UNPIVOT 命令只是查看 NEW\_DRIVER 的结果集，并处理每一行、每一列。以 UNPIVOT 运算符处理 NEW\_DRIVER 的列名为例，当遇到 ACCOUNTING 时，它会把列名 ACCOUNTING 转换为一行（位于 DNAME 列），同时它也会从 NEW\_DRIVER 中提取 ACCOUNTING 的值（等于 3），并作为 ACCOUNTING 行的一部分（位于 CNT 列）。UNPIVOT 会对 FOR 列表中指定的每一项进行这种操作，而且每项都作为一行返回。

现在，新结果集很小，它包含两列：DNAME 和 CNT，共有四行：

```

select DNAME, CNT
from (
    select [ACCOUNTING] as ACCOUNTING,
           [SALES]       as SALES,
           [RESEARCH]   as RESEARCH,
           [OPERATIONS] as OPERATIONS
    from (
        select d.dname, e.empno
        from emp e,dept d
        where e.deptno=d.deptno
        ) driver
    pivot (
        count(driver.empno)
        for driver.dname in ( [ACCOUNTING],[SALES],[RESEARCH],[OPERATIONS]
    )
    ) as empPivot
) new_driver
unpivot (cnt for dname in (ACCOUNTING,SALES,RESEARCH,OPERATIONS)
) as un_pivot

```

DNAME	CNT
ACCOUNTING	3
RESEARCH	5
SALES	6
OPERATIONS	0

## 14.3 使用 Oracle 的 MODEL 子句转换结果集

### 问题

与本章第一节相同，本节将寻求不同于传统转置技巧的另一种技巧，尝试使用 Oracle 的 MODEL 子句解决该问题。不同于 SQL Server 的 PIVOT 运算符，Oracle 的 MODEL 子

句不是为了转换结果集，事实上，更确切地说，将 MODEL 子句应用于转置是错误的行为，而且也显然不是该子句的意图所在。然而，MODEL 子句提供了解决通用问题的有趣方法。本节的具体问题是，要将以下结果集：

```
select deptno, count(*) cnt
  from emp
 group by deptno
```

DEPTNO	CNT
10	3
20	5
30	6

转换为：

D10	D20	D30
3	5	6

## 解决方案

在 MODEL 子句中，使用聚集和 CASE 表达式，其方法与传统技巧类似，主要差别是：使用数组存储聚集的值，并返回结果集的数组：

```
select max(d10) d10,
       max(d20) d20,
       max(d30) d30
  from (
select d10,d20,d30
  from ( select deptno, count(*) cnt from emp group by deptno )
 model
 dimension by(deptno d)
 measures(deptno, cnt d10, cnt d20, cnt d30)
 rules(
  d10[any] = case when deptno[cv()]=10 then d10[cv()] else 0 end,
  d20[any] = case when deptno[cv()]=20 then d20[cv()] else 0 end,
  d30[any] = case when deptno[cv()]=30 then d30[cv()] else 0 end
 )
 )
```

## 讨论

MODEL 子句是 Oracle SQL 工具箱中极其有用、功能强大的工具。如果使用了 MODEL，就会发现它的一些有用的特性，如迭代、用数组方法访问行、新建/更新结果集中的行，以及建立引用模型等等。很快就会明白，本节并没有利用 MODEL 子句提供的任何特性，但能够从多个角度看问题、以预料不到的方式使用不同特性也是不错的（如果只是研究某些特性是否比其他特性更有用，没有其他更好的理由的话）。

要理解这个解决方案，第一步是检验 FROM 子句中的内联视图。内联视图只是计算表 EMP 中每个 DEPTNO 的员工数。结果如下所示：

```
select deptno, count(*) cnt
  from emp
 group by deptno
```

DEPTNO	CNT
10	3
20	5
30	6

这个结果集是 MODEL 要处理的内容。观察 MODEL 子句，会看到三个小子句：DIMENSION BY、MEASURES 和 RULES。下面先介绍 MEASURES。

MEASURES 列表中的项就是声明用于该查询的数组，该查询使用了四个数组：DEPTNO、D10、D20 和 D30。与 SELECT 列表中的列一样，MEASURES 列表中的数组可以有别名。可以看到，四个数组中的三个都是来自内联视图的 CNT。

如果 MEASURES 列表包含数组，那么 DIMENSION BY 小子句中的项就是数组下标。思考一下：数组 D10 是 CNT 的一个别名。如果查看上述内联视图的结果集，会看到 CNT 有三个值：3、5 和 6，故在创建的 CNT 数组中就有三个元素，也就是三个整数 3、5 和 6。现在，如何从数组中单独访问这些值呢？答案是使用数组下标。DIMENSION BY 小子句中定义的下标包含三个值：10、20 和 30（来自上面的结果集）。例如，下列表达式：

```
d10[10]
```

等于 3，因为所访问的是代表 DEPTNO 10 的数组 D10 中的 CNT 值（它是 3）。

由于三个数组（D10、D20、D30）都包含 CNT 值，所以它们的结果都相同。那么，如何才能让各计数值进入正确的数组呢？答案是使用 RULES 小子句。如果查看前面给出的内联视图的结果集，会发现 DEPTNO 的值为 10、20 和 30。RULES 子句中包含 CASE 的表达式计算 DEPTNO 数组中的每个值：

- 如果值为 10，则把 DEPTNO 10 的 CNT 存储在 D10[10] 中，否则存储 0。
- 如果值为 20，则把 DEPTNO 20 的 CNT 存储在 D20[20] 中，否则存储 0。
- 如果值为 30，则把 DEPTNO 30 的 CNT 存储在 D30[30] 中，否则存储 0。

如果感觉有点儿像爱丽丝掉入兔子洞，那也不必担心，停一下，执行一下目前已讨论过的部分。下面就是已讨论部分的结果集。有时，先运行代码观察它的实际操作，然后再看文字，可能会更容易理解。运行下列代码后，它非常简单：

```
select deptno, d10,d20,d30
  from ( select deptno, count(*) cnt from emp group by deptno )
 model
 dimension by(deptno d)
 measures(deptno, cnt d10, cnt d20, cnt d30)
 rules(
   d10[any] = case when deptno[cv()]=10 then d10[cv()] else 0 end,
   d20[any] = case when deptno[cv()]=20 then d20[cv()] else 0 end,
   d30[any] = case when deptno[cv()]=30 then d30[cv()] else 0 end
 )
```

DEPTNO	D10	D20	D30
10	3	0	0

20	0	5	0
30	0	0	6

可以看到, RULES 小子句更改了每个数组的值。如果还是不能理解这些内容, 则执行如下查询, 但要注释掉 RULES 小子句的表达式:

```
select deptno, d10,d20,d30
  from ( select deptno, count(*) cnt from emp group by deptno )
 model
  dimension by(deptno d)
  measures(deptno, cnt d10, cnt d20, cnt d30)
  rules(
    /*
      d10[any] = case when deptno[cv()]=10 then d10[cv()] else 0 end,
      d20[any] = case when deptno[cv()]=20 then d20[cv()] else 0 end,
      d30[any] = case when deptno[cv()]=30 then d30[cv()] else 0 end
    */
  )
```

DEPTNO	D10	D20	D30
10	3	3	3
20	5	5	5
30	6	6	6

现在知道了, MODEL子句的结果集与内联视图相同, 唯一的差别是给 COUNT 运算分别起了别名: D10、D20 和 D30。下面的查询证明了这点:

```
select deptno, count(*) d10, count(*) d20, count(*) d30
  from emp
 group by deptno
```

DEPTNO	D10	D20	D30
10	3	3	3
20	5	5	5
30	6	6	6

因此, MODEL子句的功能如下: 获取 DEPTNO 和 CNT 的值, 并把它们放入数组中, 并确保每个数组表示一个 DEPTNO。此时, 数组 D10、D20 和 D30 都有非 0 值, 用于表示 DEPTNO 的 CNT 值。该结果集已进行了转换, 剩下的就是使用聚集函数 MAX (也可以使用 MIN 或 SUM, 在这个例子中没有差别) 返回一行:

```
select max(d10) d10,
       max(d20) d20,
       max(d30) d30
  from (
    select d10,d20,d30
      from ( select deptno, count(*) cnt from emp group by deptno )
    model
      dimension by(deptno d)
      measures(deptno, cnt d10, cnt d20, cnt d30)
      rules(
        d10[any] = case when deptno[cv()]=10 then d10[cv()] else 0 end,
        d20[any] = case when deptno[cv()]=20 then d20[cv()] else 0 end,
        d30[any] = case when deptno[cv()]=30 then d30[cv()] else 0 end
      )
  )
```

D10	D20	D30
3	5	6

## 14.4 从不固定位置提取字符串的元素

### 问题

有一个字符串字段，它包含连续的日志数据。解析整个字符串，并提取相关信息。遗憾的是，相关信息并未处于字符串的固定位置，然而，需要提取信息两端有某些特定的字符。例如，对于下列字符串：

```
xxxxxabc[867]xxx[-]xxxx[5309]xxxxx
xxxxxtime:[11271978]favnum:[4]id:[Joe]xxxxx
call:[F_GET_ROWS()]b1:[ROSEWOOD...SIR]b2:[44400002]77.90xxxxx
film:[non_marked]qq:[unit]tailpipe:[withabanana?]80sxxxxx
```

要提取方括号中的值，返回下列结果集：

FIRST_VAL	SECOND_VAL	LAST_VAL
867	-	5309
11271978	4	Joe
F_GET_ROWS()	ROSEWOOD...SIR	44400002
non_marked	unit	withabanana?

### 解决方案

尽管并不知道要提取的信息位于字符串内的确切位置，但知道它位于方括号[]中，而且有三个方括号。可以用 Oracle 的内置函数 INSTR，找到方括号的位置，然后用内置函数 SUBSTR，提取字符串中的值。视图 V 包含要进行解析的字符串，其定义如下（严格地讲，使用它只是为了增加可读性）：

```
create view V
as
select 'xxxxxabc[867]xxx[-]xxxx[5309]xxxxx' msg
  from dual
 union all
select 'xxxxxtime:[11271978]favnum:[4]id:[Joe]xxxxx' msg
  from dual
 union all
select 'call:[F_GET_ROWS()]b1:[ROSEWOOD...SIR]b2:[44400002]77.90xxxxx'
msg
  from dual
 union all
select 'film:[non_marked]qq:[unit]tailpipe:[withabanana?]80sxxxxx' msg
  from dual

1  select substr(msg,
2      instr(msg,['',1,1)+1,
3      instr(msg,']',1,1)-instr(msg,['',1,1)-1) first_val,
4      substr(msg,
5      instr(msg,['',1,2)+1,
6      instr(msg,']',1,2)-instr(msg,['',1,2)-1) second_val,
7      substr(msg,
8      instr(msg,['',-1,1)+1,
9      instr(msg,']',-1,1)-instr(msg,['',-1,1)-1) last_val
10     from V
```

## 讨论

Oracle的内置函数INSTR使这个问题相当容易解决。由于已经知道要提取的值处于方括号[]内，而且有三组方括号“[]”，那么该解决方案的第一步是就是用INSTR找到每个字符串中“[]”的位置。下面的例子返回每行中左侧方括号和右侧方括号的位置：

```
select instr(msg,['',1,1) "1st_[",
       instr(msg,']',1,1) "]"_1st",
       instr(msg,['',1,2) "2nd_[",
       instr(msg,']',1,2) "]"_2nd",
       instr(msg,['',-1,1) "3rd_[",
       instr(msg,']',-1,1) "]"_3rd"
from V
```

1st_[ ]_1st	2nd_[ ]_2nd	3rd_[ ]_3rd
9 13	17 19	24 29
11 20	28 30	34 38
6 19	23 38	42 51
6 17	21 26	36 49

至此，困难的工作已完成。剩下的就是用数字位置作为SUBSTR的参数，并解析MSG在这些位置的信息。注意，在完整的解决方案中，对INSTR返回的值进行了+1和-1计算，这样最终结果集就不会包含左方括号“[”。下面给出的解决方案没有对INSTR的返回值进行加1和减1操作，注意它都以方括号开始：

```
select substr(msg,
             instr(msg,['',1,1),
             instr(msg,']',1,1)-instr(msg,['',1,1)) first_val,
       substr(msg,
             instr(msg,['',1,2),
             instr(msg,']',1,2)-instr(msg,['',1,2)) second_val,
       substr(msg,
             instr(msg,['',-1,1),
             instr(msg,']',-1,1)-instr(msg,['',-1,1)) last_val
from V
```

FIRST_VAL	SECOND_VAL	LAST_VAL
[867	[-	[5309
[11271978	[4	[Joe
[F_GET_ROWS()	[ROSEWOOD...SIR	[44400002
[non_marked	[unit	[withabanana?

从上面的结果集可以看出，返回的信息包含左侧方括号。读者可能会想：“好吧，给INSTR加1，就会使前导方括号消失。那为什么减1呢？”原因是：如果只把位置加1，其他相关的地方不减，就会将右方括号包含进来，如下所示：

```
select substr(msg,
             instr(msg,['',1,1)+1,
             instr(msg,']',1,1)-instr(msg,['',1,1)) first_val,
       substr(msg,
             instr(msg,['',1,2)+1,
             instr(msg,']',1,2)-instr(msg,['',1,2)) second_val,
       substr(msg,
             instr(msg,['',-1,1)+1,
             instr(msg,']',-1,1)-instr(msg,['',-1,1)) last_val
from V
```



FIRST_VAL	SECOND_VAL	LAST_VAL
867]	-]	5309]
11271978]	4]	Joe]
F_GET_ROWS()]	ROSEWOOD...SIR]	44400002]
non_marked]	unit]	withabanana?]

至此，应该明白：要确保不包含方括号，必须对起始下标加1，对结束下标减1。

## 14.5 求一年包含的天数（Oracle 的另一种解决方案） 问题

找到一年包含的天数。

---

注意：针对第9章9.2节提出的“确定一年的天数”问题，本节将介绍另一种解决方案。该解决方案专用于 Oracle。

---

### 解决方案

使用 TO\_CHAR 函数，把一年的最后一个日期设置为 3 位数字的年份日期：

```

1 select 'Days in 2005: '||
2      to_char(add_months(trunc(sysdate,'y'),12)-1,'DDD')
3      as report
4  from dual
5 union all
6 select 'Days in 2004: '||
7      to_char(add_months(trunc(
8          to_date('01-SEP-2004'),'y'),12)-1,'DDD')
9  from dual

REPORT
-----
Days in 2005: 365
Days in 2004: 366

```

### 讨论

首先，使用 TRUNC 函数，返回给定日期所在年份的第一天，如下所示：

```

select trunc(to_date('01-SEP-2004'),'y')
   from dual

TRUNC(TO_DA
-----
01-JAN-2004

```

下一步，使用 ADD\_MONTHS，给截断的日期加 1 年（12 个月）。再减 1 天，就获得了原始日期所在年份的最后一天：

```

select add_months(
    trunc(to_date('01-SEP-2004'),'y'),
    12) before_subtraction,
    add_months(
    trunc(to_date('01-SEP-2004'),'y'),

```

```

        12)-1 after_subtraction
    from dual
BEFORE_SUBT AFTER_SUBTR
-----
01-JAN-2005 31-DEC-2004

```

现在, 已经找到了一年的最后一天, 只要使用 TO\_CHAR, 返回一个 3 位数的年份日期, 用于表示最后一天是当中年的第几天 (第一天、第五十天, 等等):

```

select to_char(
    add_months(
        trunc(to_date('01-SEP-2004'),'y'),
        12)-1,'DDD') num_days_in_2004
    from dual
NUM
---
366

```

## 14.6 搜索字母数字混合的字符串

### 问题

某列包含字母数字混合的数据, 要返回那些既包含字母字符又包含数字字符的行, 换句话说, 如果字符串只包含数字或只包含字母, 就不返回它, 返回的值应该是字符和数字的混合。对于下列数据:

```

STRINGS
-----
1010 switch
333
3453430278
ClassSummary
findRow 55
threes

```

其最终结果集应该包含那些既有字母又有数字的行:

```

STRINGS
-----
1010 switch
findRow 55

```

### 解决方案

使用内置函数 TRANSLATE, 分别把所有字母和所有数字转换为特定字符, 然后, 只保留那些包含两种特定字符至少各一个的字符串。该解决方案使用的是 Oracle 语法, 但 DB2 和 PostgreSQL 都支持 TRANSLATE, 所以要想在这些平台上实现该解决方案, 只需进行简单修改:

```

with v as (
select 'ClassSummary' strings from dual union
select '3453430278'      from dual union
select 'findRow 55'     from dual union
select '1010 switch'    from dual union
select '333'            from dual union

```

```

select 'threes'          from dual
)
select strings
  from (
select strings,
       translate(
         strings,
         'abcdefghijklmnopqrstuvwxyz0123456789',
         rpad('#',26,'#')||rpad('*',10,'*')) translated
  from v
) x
where instr(translated,'#') > 0
       and instr(translated,'*') > 0

```

注意：还有另一种方案，不用 WITH 子句，而使用内联视图，或者就创建一个视图。

## 讨论

使用 TRANSLATE 函数，问题将非常容易解决。要使用 TRANSLATE，第一步用井号 (#) 字符和星号 (\*) 字符分别代替所有字母和所有数字。中间结果（内联视图 X）如下：

```

with v as (
select 'ClassSummary' strings from dual union
select '3453430278'      from dual union
select 'findRow 55'      from dual union
select '1010 switch'     from dual union
select '333'              from dual union
select 'threes'          from dual
)
select strings,
       translate(
         strings,
         'abcdefghijklmnopqrstuvwxyz0123456789',
         rpad('#',26,'#')||rpad('*',10,'*')) translated
  from v

```

STRINGS	TRANSLATED
1010 switch	**** #####
333	***
3453430278	*****
ClassSummary	C####S#####
findRow 55	####R## **
threes	#####

至此，只需要保留至少包含一个 “#” 和 “\*” 的行就可以了。用函数 INSTR 确定字符串中是否包含 “#” 和 “\*”。如果存在这两个字符，那么返回的值将大于 0。为清晰起见，下面既返回了最终需要的字符串，也返回了对它们做转换后的中间值：

```

with v as (
select 'ClassSummary' strings from dual union
select '3453430278'      from dual union
select 'findRow 55'      from dual union
select '1010 switch'     from dual union
select '333'              from dual union
select 'threes'          from dual
)
select strings, translated
  from (
select strings,

```

```

        translate(
            strings,
            'abcdefghijklmnopqrstuvwxyz0123456789',
            rpad('#',26,'#')||rpad('*',10,'*')) translated
    from v
    )
    where instr(translated,'#') > 0
        and instr(translated,'*') > 0

STRINGS          TRANSLATED
-----
1010 switch      **** #####
findRow 55       #####R## **

```

## 14.7 使用 Oracle 把整数转换为二进制数

### 问题

在 Oracle 系统中，把整数转换为二进制数。例如，以二进制格式返回表 EMP 中的所有工资，部分结果集如下：

ENAME	SAL	SAL_BINARY
SMITH	800	1100100000
ALLEN	1600	11001000000
WARD	1250	10011100010
JONES	2975	101110011111
MARTIN	1250	10011100010
BLAKE	2850	101100100010
CLARK	2450	100110010010
SCOTT	3000	101110111000
KING	5000	1001110001000
TURNER	1500	10111011100
ADAMS	1100	10001001100
JAMES	950	1110110110
FORD	3000	101110111000
MILLER	1300	10100010100

### 解决方案

这个解决方案使用了 MODEL 子句，所以它必须在 Oracle Database 10g（或更高版本）系统下才能正常运行。由于 MODEL 具有迭代功能，而且也提供了以数组方式访问行的功能，因此选用它是很自然的事（如果一定要用 SQL 解决该问题，那么选用存储函数会更合适）。与本书中的其他解决方案一样，即使不能将这段代码用于实际的应用程序中，还是要学习这种技巧。MODEL 子句既能够完成过程任务，而且依然保留 SQL 的基于集合的特性和功能。甚至有人说：“我从来不在 SQL 中做这件事”，确实如此，我决不建议读者应该怎么样或不应该怎么样，只是提醒读者关注其中的技巧，以便把它用于更“实用的”应用程序中。

下面的解决方案返回表 EMP 中的所有 ENAME 和 SAL，同时在标量子查询中调用了 MODEL 子句（它就像一种孤立函数，只是接收表 EMP 的输入，并处理它，然后返回一个值）：

```
1 select ename,
```

```

2      sal,
3      (
4      select bin
5      from dual
6      model
7      dimension by ( 0 attr )
8      measures ( sal num,
9                  cast(null as varchar2(30)) bin,
10                 '0123456789ABCDEF' hex
11              )
12      rules iterate (10000) until (num[0] <= 0) {
13          bin[0] = substr(hex[cv()],mod(num[cv()],2)+1,1)||bin[cv()],
14          num[0] = trunc(num[cv()]/2)
15      }
16      ) sal_binary
17 from emp

```

## 讨论

在“解决方案”部分曾经提到：对这种问题采用存储函数可能更合适，事实上，本节的灵感就来自函数。本节是对TO\_BASE函数的改编，该函数是由Oracle公司的Tom Kyte编写的。与本书的其他章节一样，也可以决定不用它，即使不采用本方案，了解MODEL子句的一些特征（如迭代、以数组方式访问行）也是相当不错的。

为了更容易解释，这里对包含MODEL子句的子查询稍做修改。下面的代码就是该解决方案的子查询，只是把它固定为返回2的二进制值：

```

select bin
from dual
model
dimension by ( 0 attr )
measures ( 2 num,
           cast(null as varchar2(30)) bin,
           '0123456789ABCDEF' hex
         )
rules iterate (10000) until (num[0] <= 0) {
    bin[0] = substr (hex[cv()],mod(num[cv()],2)+1,1)||bin[cv()],
    num[0] = trunc(num[cv()]/2)
}
)
BIN
-----
10

```

下列查询输出的是上述查询的RULES中所定义的迭代执行一次的返回值：

```

select 2 start_val,
       '0123456789ABCDEF' hex,
       substr('0123456789ABCDEF',mod(2,2)+1,1) ||
       cast(null as varchar2(30)) bin,
       trunc(2/2) num
from dual

```

START_VAL	HEX	BIN	NUM
2	0123456789ABCDEF	0	1

START\_VAL 表示要转换为二进制的数字，在这个例子中，它是2。BIN 值是对`0123456789ABCDEF'子串进行操作的结果（原始解决方案中的HEX）。NUM值用于判断何时退出循环。

从上面的结果集可以看出，第一次执行循环时，BIN是0，NUM是1。由于NUM大于0，所以会继续进行循环迭代。下列 SQL 语句展示了执行下一次重复的结果：

```
select num start_val,
       substr('0123456789ABCDEF',mod(1,2)+1,1) || bin bin,
       trunc(1/2) num
from (
select 2 start_val,
       '0123456789ABCDEF' hex,
       substr('0123456789ABCDEF',mod(2,2)+1,1) ||
       cast(null as varchar2(30)) bin,
       trunc(2/2) num
from dual
)
```

START_VAL	BIN	NUM
1	10	0

这次循环之后，对 HEX 子串操作的结果将返回 1，而且把 BIN 的前一个值 0 跟它连接起来。现在，NUM 是 0，因此，它是最后一次迭代，并返回“10”，这是数字 2 的二进制表示法。一旦理解了这些内容，就可以不用理会 MODEL 子句中的迭代，而去关注如何一行一行使用规则，得到最终结果集，如下所示：

```
select 2 orig_val, num, bin
from dual
model
dimension by ( 0 attr )
measures ( 2 num,
           cast(null as varchar2(30)) bin,
           '0123456789ABCDEF' hex
)
rules (
  bin[0] = substr (hex[cv()],mod(num[cv()],2)+1,1)||bin[cv()],
  num[0] = trunc(num[cv()]/2),
  bin[1] = substr (hex[0],mod(num[0],2)+1,1)||bin[0],
  num[1] = trunc(num[0]/2)
)
```

ORIG_VAL	NUM	BIN
2	1	0
2	0	10

## 14.8 转置已分等级的结果集

### 问题

给表中的值分等级，然后把结果集转置为三列，其思想是将最高的 3 档作为一列、次高的 3 档作一列，其余作一列。例如，想给表 EMP 中的员工按 SAL 分等级，然后把结果集转置为三列。其结果集如下所示：

TOP_3	NEXT_3	REST
KING (5000)	BLAKE (2850)	TURNER (1500)
FORD (3000)	CLARK (2450)	MILLER (1300)
SCOTT (3000)	ALLEN (1600)	MARTIN (1250)
JONES (2975)		WARD (1250)
		ADAMS (1100)

JAMES (950)  
SMITH (800)

## 解决方案

这个解决方案的关键是先使用窗口函数 DENSE\_RANK OVER，按 SAL 给员工分等级，同时允许捆绑。使用 DENSE\_RANK OVER，可以很容易地看到最高的三档工资、接下来的三档工资，以及其余的所有工资。下一步，使用窗口函数 ROW\_NUMBER OVER，给组内的每个员工分等级（最高组、次高组，及其余组）。之后，只要进行标准转换，或使用平台支持的内置字符串函数，美化结果即可。下列解决方案采用了 Oracle 语法。由于 DB2 和 SQL Server 2005 都支持窗口函数，所以，修改适用于这些平台的解决方案是非常简单的：

```

1  select max(case grp when 1 then rpad(ename,6) ||
2             ' (|| sal ||)' end) top_3,
3             max(case grp when 2 then rpad(ename,6) ||
4             ' (|| sal ||)' end) next_3,
5             max(case grp when 3 then rpad(ename,6) ||
6             ' (|| sal ||)' end) rest
7  from (
8  select ename,
9         sal,
10        rnk,
11        case when rnk <= 3 then 1
12              when rnk <= 6 then 2
13              else 3
14        end grp,
15        row_number()over (
16          partition by case when rnk <= 3 then 1
17                        when rnk <= 6 then 2
18                        else 3
19          end
20          order by sal desc, ename
21        ) grp_rnk
22  from (
23  select ename,
24         sal,
25         dense_rank()over(order by sal desc) rnk
26  from emp
27  ) x
28  ) y
29  group by grp_rnk

```

## 讨论

本节是个完美的例子，它说明了利用窗口函数，一点代码就能够实现多少功能。该解决方案看起来可能有点棘手，但如果从内到外分解开，就会惊讶于它多么简单。下面先执行内联视图 X：

```

select ename,
       sal,
       dense_rank()over(order by sal desc) rnk
from emp

```

ENAME	SAL	RNK
KING	5000	1
SCOTT	3000	2

FORD	3000	2
JONES	2975	3
BLAKE	2850	4
CLARK	2450	5
ALLEN	1600	6
TURNER	1500	7
MILLER	1300	8
WARD	1250	9
MARTIN	1250	9
ADAMS	1100	10
JAMES	950	11
SMITH	800	12

从上面的结果集可以看出，内联视图 X 只是按 SAL 给员工分等级，同时允许捆绑（因为该解决方案用 DENSE\_RANK 而不是 RANK，所以捆绑之间没有间隔）。下一步，用 CASE 表达式判断 DENSE\_RANK 求出的等级，并据此对内联视图 X 的结果集创建组。此外，= 用窗口函数 ROW\_NUMBER OVER，按 SAL 给组（是使用 CASE 表达式创建的）内的员工分等级。所有这些都在内联视图 Y 中进行，如下所示：

```

select ename,
       sal,
       rnk,
       case when rnk <= 3 then 1
            when rnk <= 6 then 2
            else 3
       end grp,
       row_number()over (
         partition by case when rnk <= 3 then 1
                      when rnk <= 6 then 2
                      else 3
         end
         order by sal desc, ename
       ) grp_rnk
from (
select ename,
       sal,
       dense_rank()over(order by sal desc) rnk
from emp
) x

```

ENAME	SAL	RNK	GRP	GRP_RNK
KING	5000	1	1	1
FORD	3000	2	1	2
SCOTT	3000	2	1	3
JONES	2975	3	1	4
BLAKE	2850	4	2	1
CLARK	2450	5	2	2
ALLEN	1600	6	2	3
TURNER	1500	7	3	1
MILLER	1300	8	3	2
MARTIN	1250	9	3	3
WARD	1250	9	3	4
ADAMS	1100	10	3	5
JAMES	950	11	3	6
SMITH	800	12	3	7

现在，该查询已初具雏形，如果读者是从头开始（内联视图 X）一步步跟下来，就会觉得其实并不复杂。至此，查询返回每个员工，他的 SAL、他的 RNK（表示他的 SAL 在所有员工间的等级）、他的 GRP（根据其 SAL 在员工所属组），最后一个 GRP\_RANK，在 GRP 内的等级（基于 SAL）。



这时，对ENAME进行传统转置，并使用Oracle字符串连接运算符“||”把它跟SAL连接起来。函数RPAD确保圆括号内的数字值整齐排列。最后，对GRP\_RNK使用GROUP BY，以确保在结果集中显示每个员工。最终结果集如下：

```
select max(case grp when 1 then rpad(ename,6) ||
         ' (|| sal ||)' end) top_3,
       max(case grp when 2 then rpad(ename,6) ||
         ' (|| sal ||)' end) next_3,
       max(case grp when 3 then rpad(ename,6) ||
         ' (|| sal ||)' end) rest
  from (
select ename,
       sal,
       rnk,
       case when rnk <= 3 then 1
            when rnk <= 6 then 2
            else 3
       end grp,
       row_number()over (
         partition by case when rnk <= 3 then 1
                        when rnk <= 6 then 2
                        else 3
         end
         order by sal desc, ename
       ) grp_rnk
  from (
select ename,
       sal,
       dense_rank()over(order by sal desc) rnk
  from emp
  ) x
  ) y
 group by grp_rnk
```

TOP_3	NEXT_3	REST
KING (5000)	BLAKE (2850)	TURNER (1500)
FORD (3000)	CLARK (2450)	MILLER (1300)
SCOTT (3000)	ALLEN (1600)	MARTIN (1250)
JONES (2975)		WARD (1250)
		ADAMS (1100)
		JAMES (950)
		SMITH (800)

如果检验所有步骤的查询，会发现，整个过程只访问了一次表EMP。对于窗口函数，需要关注在对数据的一次扫描中能够做多少工作。不需要进行自联接，也不需要临时表，只是获得想要的行，其他工作都交给窗口函数。仅在内联视图X中需要访问EMP，从那儿开始，就只是将结果集组织成所需要的信息。想一想，创建这种类型的报表，而且只访问表一次，这意味着什么样的性能！非常酷！

## 14.9 给两次转置的结果集增加列头

### 问题

把两个结果集叠在一起，然后把它们转置为两列，另外，还要为每列加一个“标题”。例如，有两个表，它们包含公司中有关员工的信息，这些员工在不同地区从事开发工作（也即研究和应用）：

```
select * from it_research
```

```
DEPTNO ENAME
-----
100 HOPKINS
100 JONES
100 TONEY
200 MORALES
200 P.WHITAKER
200 MARCIANO
200 ROBINSON
300 LACY
300 WRIGHT
300 J.TAYLOR
```

```
select * from it_apps
```

```
DEPTNO ENAME
-----
400 CORRALES
400 MAYWEATHER
400 CASTILLO
400 MARQUEZ
400 MOSLEY
500 GATTI
500 CALZAGHE
600 LAMOTTA
600 HAGLER
600 HEARNS
600 FRAZIER
700 GUINN
700 JUDAH
700 MARGARITO
```

要创建一个报表，它分两栏列出两个表中的员工。对于每一列，都返回 DEPTNO 及 ENAME。最后，返回下列结果集：

RESEARCH	APPS
-----	-----
100	400
JONES	MAYWEATHER
TONEY	CASTILLO
HOPKINS	MARQUEZ
200	MOSLEY
P.WHITAKER	CORRALES
MARCIANO	500
ROBINSON	CALZAGHE
MORALES	GATTI
300	600
WRIGHT	HAGLER
J.TAYLOR	HEARNS
LACY	FRAZIER
	LAMOTTA
	700
	JUDAH
	MARGARITO
	GUINN

## 解决方案

本解决方案只需要一个简单的堆叠及转置（合并后转置）并且再“拧”一次：DEPTNO 一定在每个员工的ENAME之前。这里的技巧采用了笛卡儿积为每个DEPTNO生成附加行，这样才有足够的行显示所有员工和DEPTNO。该解决方案采用 Oracle 语法，但由于

DB2 也支持计算移动窗口的窗口函数（框架子句），把这个解决方案转换为 DB2 平台的方案是非常容易的。因为 IT\_RESEARCH 和 IT\_APPS 表只用于解决本节的问题，所以该解决方案也列出了它们的创建语句：

```
create table IT_research (deptno number, ename varchar2(20))

insert into IT_research values (100,'HOPKINS')
insert into IT_research values (100,'JONES')
insert into IT_research values (100,'TONEY')
insert into IT_research values (200,'MORALES')
insert into IT_research values (200,'P.WHITAKER')
insert into IT_research values (200,'MARCIANO')
insert into IT_research values (200,'ROBINSON')
insert into IT_research values (300,'LACY')
insert into IT_research values (300,'WRIGHT')
insert into IT_research values (300,'J.TAYLOR')
```

```
create table IT_apps (deptno number, ename varchar2(20))

insert into IT_apps values (400,'CORRALES')
insert into IT_apps values (400,'MAYWEATHER')
insert into IT_apps values (400,'CASTILLO')
insert into IT_apps values (400,'MARQUEZ')
insert into IT_apps values (400,'MOSLEY')
insert into IT_apps values (500,'GATTI')
insert into IT_apps values (500,'CALZAGHE')
insert into IT_apps values (600,'LAMOTTA')
insert into IT_apps values (600,'HAGLER')
insert into IT_apps values (600,'HEARNS')
insert into IT_apps values (600,'FRAZIER')
insert into IT_apps values (700,'GUINN')
insert into IT_apps values (700,'JUDAH')
insert into IT_apps values (700,'MARGARITO')

1  select max(decode(flag2,0,it_dept)) research,
2  max(decode(flag2,1,it_dept)) apps
3  from (
4  select sum(flag1)over(partition by flag2
5  order by flag1,rownum) flag,
6  it_dept, flag2
7  from (
8  select 1 flag1, 0 flag2,
9  decode(rn,1,to_char(deptno),' ||ename) it_dept
10 from (
11 select x.*, y.id,
12 row_number()over(partition by x.deptno order by y.id) rn
13 from (
14 select deptno,
15 ename,
16 count(*)over(partition by deptno) cnt
17 from it_research
18 ) x,
19 (select level id from dual connect by level <= 2) y
20 )
21 where rn <= cnt+1
22 union all
23 select 1 flag1, 1 flag2,
24 decode(rn,1,to_char(deptno),' ||ename) it_dept
25 from (
26 select x.*, y.id,
27 row_number()over(partition by x.deptno order by y.id) rn
28 from (
29 select deptno,
30 ename,
31 count(*)over(partition by deptno) cnt
```

```

32   from it_apps
33   ) x,
34   (select level id from dual connect by level <= 2) y
35   )
36   where rn <= cnt+1
37   ) tmp1
38   ) tmp2
39   group by flag

```

## 讨论

与其他数据仓库/报表类型查询一样，这个解决方案看起来也有点儿令人费解，但如果分解开来，就会明白它只是一个堆叠、转置及笛卡儿扭转（在岩石，用把小伞）。要分解这个查询，应该先逐个运行 UNION ALL 的每一部分，然后再连起来进行转置。先从 UNION ALL 的下半部开始：

```

select 1 flag1, 1 flag2,
       decode(rn,1,to_char(deptno),' '||ename) it_dept
  from (
select x.*, y.id,
       row_number()over(partition by x.deptno order by y.id) rn
  from (
select deptno,
       ename,
       count(*)over(partition by deptno) cnt
  from it_apps
  ) x,
  (select level id from dual connect by level <= 2) y
  ) z
 where rn <= cnt+1

```

FLAG1	FLAG2	IT_DEPT
1	1	400
1	1	MAYWEATHER
1	1	CASTILLO
1	1	MARQUEZ
1	1	MOSLEY
1	1	CORRALES
1	1	500
1	1	CALZAGHE
1	1	GATTI
1	1	600
1	1	HAGLER
1	1	HEARNS
1	1	FRAZIER
1	1	LAMOTTA
1	1	700
1	1	JUDAH
1	1	MARGARITO
1	1	GUINN

现在，检验结果集是如何组成的。把上面的查询分解成最简单的组件，内联视图 X 只是返回所有 ENAME 和 DEPTNO，也返回了表 IT\_APPS 中每个 DEPTNO 的员工数。其结果如下所示：

```

select deptno deptno,
       ename,
       count(*)over(partition by deptno) cnt
  from it_apps

```

DEPTNO	ENAME	CNT
400	CORRALES	5
400	MAYWEATHER	5
400	CASTILLO	5
400	MARQUEZ	5
400	MOSLEY	5
500	GATTI	2
500	CALZAGHE	2
600	LAMOTTA	4
600	HAGLER	4
600	HEARNS	4
600	FRAZIER	4
700	GUINN	3
700	JUDAH	3
700	MARGARITO	3

下一步，使用 CONNECT BY，创建内联视图 X 返回的行及 DUAL 生成的两行之间的笛卡儿积。这种操作的结果如下所示：

```
select *
  from (
select deptno deptno,
       ename,
       count(*)over(partition by deptno) cnt
  from it_apps
) x,
  (select level id from dual connect by level <= 2) y
 order by 2
```

DEPTNO	ENAME	CNT	ID
500	CALZAGHE	2	1
500	CALZAGHE	2	2
400	CASTILLO	5	1
400	CASTILLO	5	2
400	CORRALES	5	1
400	CORRALES	5	2
600	FRAZIER	4	1
600	FRAZIER	4	2
500	GATTI	2	1
500	GATTI	2	2
700	GUINN	3	1
700	GUINN	3	2
600	HAGLER	4	1
600	HAGLER	4	2
600	HEARNS	4	1
600	HEARNS	4	2
700	JUDAH	3	1
700	JUDAH	3	2
600	LAMOTTA	4	1
600	LAMOTTA	4	2
700	MARGARITO	3	1
700	MARGARITO	3	2
400	MARQUEZ	5	1
400	MARQUEZ	5	2
400	MAYWEATHER	5	1
400	MAYWEATHER	5	2
400	MOSLEY	5	1
400	MOSLEY	5	2

从这些结果可以看到，对于内联视图 X 的每一行都返回了两次，这是它与内联视图 Y 笛卡儿积的结果。不久就会更清楚为什么需要笛卡儿。下一步是获取当前结果集，并按 ID（笛卡儿积返回的 ID 值为 1 或 2）给 DEPTNO 内的所有员工分等级。下列查询的输出显示了这种结果：

```

select x.*, y.id,
       row_number()over(partition by x.deptno order by y.id) rn
  from (
select deptno deptno,
       ename,
       count(*)over(partition by deptno) cnt
  from it_apps
) x,
     (select level id from dual connect by level <= 2) y

```

DEPTNO	ENAME	CNT	ID	RN
400	CORRALES	5	1	1
400	MAYWEATHER	5	1	2
400	CASTILLO	5	1	3
400	MARQUEZ	5	1	4
400	MOSLEY	5	1	5
400	CORRALES	5	2	6
400	MOSLEY	5	2	7
400	MAYWEATHER	5	2	8
400	CASTILLO	5	2	9
400	MARQUEZ	5	2	10
500	GATTI	2	1	1
500	CALZAGHE	2	1	2
500	GATTI	2	2	3
500	CALZAGHE	2	2	4
600	LAMOTTA	4	1	1
600	HAGLER	4	1	2
600	HEARNS	4	1	3
600	FRAZIER	4	1	4
600	LAMOTTA	4	2	5
600	HAGLER	4	2	6
600	FRAZIER	4	2	7
600	HEARNS	4	2	8
700	GUINN	3	1	1
700	JUDAH	3	1	2
700	MARGARITO	3	1	3
700	GUINN	3	2	4
700	JUDAH	3	2	5
700	MARGARITO	3	2	6

每个员工都分了等级，他的复本也分了等级。结果集包含表 IT\_APP 中所有员工的复本，也包含它们在 DEPTNO 内的等级。需要生成这些额外行的原因是：结果集中需要一个缝隙，以便在 ENAME 列放入 DEPTNO。如果把 IT\_APPS 与一行表进行笛卡儿联接，就不会得到额外行（因为某个表的基数  $X1 = \text{该表的基数}$ ）。

下一步，获取返回的结果，并转置结果集，这样，返回的所有 ENAMES 都会处于一行，而且 DEPTNO 在这些 ENAMES 之前。下列查询显示了这种操作的结果：

```

select 1 flag1, 1 flag2,
       decode(rn,1,to_char(deptno),' '||ename) it_dept
  from (
select x.*, y.id,
       row_number()over(partition by x.deptno order by y.id) rn
  from (
select deptno deptno,
       ename,
       count(*)over(partition by deptno) cnt
  from it_apps
) x,
     (select level id from dual connect by level <= 2) y
) z
where rn <= cnt+1

```

FLAG1	FLAG2	IT_DEPT
1	1	400
1	1	MAYWEATHER
1	1	CASTILLO
1	1	MARQUEZ
1	1	MOSLEY
1	1	CORRALES
1	1	500
1	1	CALZAGHE
1	1	GATTI
1	1	600
1	1	HAGLER
1	1	HEARNS
1	1	FRAZIER
1	1	LAMOTTA
1	1	700
1	1	JUDAH
1	1	MARGARITO
1	1	GUINN

对于 FLAG1 和 FLAG2，稍后才会用到，现在可以不管它们，将注意力集中于 IT\_DEPT 行。为每个 DEPTNO 返回的行数是 CNT\*2，但需要的行数是 CNT+1，WHERE 子句中的筛选就是这样的。RN 是每个员工的等级，保留下来的行是已经等级小于等于 CNT+1 的行，也就是每个 DEPTNO 中的所有员工再加 1（这个额外的员工是 DEPTNO 中先分等级的员工），DEPTNO 将放入这个附加行。使用 DECODE（一个比较老的 Oracle 函数，它的功能与 CASE 表达式差不多）判断 RN 的值，把 DEPTNO 值放入结果集。处于第一个位置的员工（基于 RN 值）依然存在于结果集中，但他现在成了每个 DEPTNO 中的最后一位（因为次序没什么关系，这不是问题）。至此，介绍完了 UNION ALL 的下半部。

UNION ALL 的上半部与下半部的处理方式相同，因此不需要再进行解释了。下面只查看堆叠以后的结果集：

```
select 1 flag1, 0 flag2,
       decode(rn,1,to_char(deptno),' '||ename) it_dept
  from (
select x.*, y.id,
       row_number()over(partition by x.deptno order by y.id) rn
  from (
select deptno,
       ename,
       count(*)over(partition by deptno) cnt
  from it_research
    ) x,
    (select level id from dual connect by level <= 2) y
  where rn <= cnt+1
 union all
select 1 flag1, 1 flag2,
       decode(rn,1,to_char(deptno),' '||ename) it_dept
  from (
select x.*, y.id,
       row_number()over(partition by x.deptno order by y.id) rn
  from (
select deptno deptno,
       ename,
       count(*)over(partition by deptno) cnt
  from it_apps
    ) x,
```

```

        (select level id from dual connect by level <= 2) y
      )
  where rn <= cnt+1

```

FLAG1	FLAG2	IT_DEPT
1	0	100
1	0	JONES
1	0	TONEY
1	0	HOPKINS
1	0	200
1	0	P.WHITAKER
1	0	MARCIANO
1	0	ROBINSON
1	0	MORALES
1	0	300
1	0	WRIGHT
1	0	J.TAYLOR
1	0	LACY
1	1	400
1	1	MAYWEATHER
1	1	CASTILLO
1	1	MARQUEZ
1	1	MOSLEY
1	1	CORRALES
1	1	500
1	1	CALZAGHE
1	1	GATTI
1	1	600
1	1	HAGLER
1	1	HEARNS
1	1	FRAZIER
1	1	LAMOTTA
1	1	700
1	1	JUDAH
1	1	MARGARITO
1	1	GUINN

至此，对于FLAG1的用途还不清楚，但可以看到，FLAG2用于识别行来自UNION ALL的哪部分（0代表上半部，1代表下半部）。

下一步，把结果集包入内联视图，并在FLAG1上创建累加和（最终，它的用途暴露出来了！），它分别在堆叠两部分内充当各行的等级，等级的结果（累加和）如下所示：

```

select sum(flag1)over(partition by flag2
                      order by flag1,rownum) flag,
       it_dept, flag2
  from (
select 1 flag1, 0 flag2,
       decode(rn,1,to_char(deptno),' '||ename) it_dept
  from (
select x.*, y.id,
       row_number()over(partition by x.deptno order by y.id) rn
  from (
select deptno,
       ename,
       count(*)over(partition by deptno) cnt
  from it_research
  ) x,
       (select level id from dual connect by level <= 2) y
  )
  where rn <= cnt+1
 union all
select 1 flag1, 1 flag2,

```



```

        decode(rn,1,to_char(deptno),' '||ename) it_dept
    from (
select x.*, y.id,
       row_number()over(partition by x.deptno order by y.id) rn
    from (
select deptno deptno,
       ename,
       count(*)over(partition by deptno) cnt
    from it_apps
       ) x,
       (select level id from dual connect by level <= 2) y
    where rn <= cnt+1
       ) tmp1

```

FLAG	IT_DEPT	FLAG2
1	100	0
2	JONES	0
3	TONEY	0
4	HOPKINS	0
5	200	0
6	P.WHITAKER	0
7	MARCIANO	0
8	ROBINSON	0
9	MORALES	0
10	300	0
11	WRIGHT	0
12	J.TAYLOR	0
13	LACY	0
1	400	1
2	MAYWEATHER	1
3	CASTILLO	1
4	MARQUEZ	1
5	MOSLEY	1
6	CORRALES	1
7	500	1
8	CALZAGHE	1
9	GATTI	1
10	600	1
11	HAGLER	1
12	HEARNS	1
13	FRAZIER	1
14	LAMOTTA	1
15	700	1
16	JUDAH	1
17	MARGARITO	1
18	GUINN	1

最后一步(终于要结束了!),是在FLAG2上转置TMP1返回的值,同时按FLAG1(TMP1中生成的累加和)分组。把TMP1生成的结果包入内联视图中,并进行转置(包入TMP2内联视图中)。最终解决方案及结果集如下所示:

```

select max(decode(flag2,0,it_dept)) research,
       max(decode(flag2,1,it_dept)) apps
    from (
select sum(flag1)over(partition by flag2
                      order by flag1,rownum) flag,
       it_dept, flag2
    from (
select 1 flag1, 0 flag2,
       decode(rn,1,to_char(deptno),' '||ename) it_dept
    from (
select x.*, y.id,
       row_number()over(partition by x.deptno order by y.id) rn

```

```

        from (
select deptno,
       ename,
       count(*)over(partition by deptno) cnt
  from it_research
       ) x,
       (select level id from dual connect by level <= 2) y
       )
  where rn <= cnt+1
 union all
select 1 flag1, 1 flag2,
       decode(rn,1,to_char(deptno),' '||ename) it_dept
  from (
select x.*, y.id,
       row_number()over(partition by x.deptno order by y.id) rn
  from (
select deptno deptno,
       ename,
       count(*)over(partition by deptno) cnt
  from it_apps
       ) x,
       (select level id from dual connect by level <= 2) y
       )
  where rn <= cnt+1
       ) tmp1
       ) tmp2
 group by flag

```

RESEARCH	APPS
-----	-----
100	400
JONES	MAYWEATHER
TONEY	CASTILLO
HOPKINS	MARQUEZ
200	MOSLEY
P.WHITAKER	CORRALES
MARCIANO	500
ROBINSON	CALZAGHE
MORALES	GATTI
300	600
WRIGHT	HAGLER
J.TAYLOR	HEARNS
LACY	FRAZIER
	LAMOTTA
	700
	JUDAH
	MARGARITO
	GUINN

## 14.10 在 Oracle 中把标量子查询转换为复合子查询问题

绕开标量子查询只能返回一个值的限制。例如，试图执行下列查询：

```

select e.deptno,
       e.ename,
       e.sal,
       (select d.dname,d.loc,sysdate today
        from dept d
        where e.deptno=d.deptno)
  from emp e

```

会收到一个错误，原因是 SELECT 列表中的子查询只能返回一个值。

## 解决方案

诚然，这个问题很不切实际，因为表 EMP 和 DEPT 之间做个简单联接就可以从 DEPT 中任意返回想要的值。然而，这里的关键是集中于技巧的讨论，以及了解如何把它应用于可能有用的场合。当在 SELECT 内加 SELECT（标量子查询）时，要绕开其只能返回单个值的限制，关键是利用 Oracle 的对象类型。可以给一个对象定义几个属性，然后把它当作单个实体进行处理，或单独引用每个元素。实际上，根本没有绕过规则，只是返回一个值、一个包含很多属性的对象。

该解决方案使用了下列对象类型：

```
create type generic_obj
as object (
    val1 varchar2(10),
    val2 varchar2(10),
    val3 date
);
```

如果采用了这种类型，则可以执行下列查询：

```
1 select x.deptno,
2       x.ename,
3       x.multival.val1 dname,
4       x.multival.val2 loc,
5       x.multival.val3 today
6   from (
7       select e.deptno,
8              e.ename,
9              e.sal,
10             (select generic_obj(d.dname,d.loc,sysdate+1)
11              from dept d
12              where e.deptno=d.deptno) multival
13   from emp e
14   ) x
```

DEPTNO	ENAME	DNAME	LOC	TODAY
20	SMITH	RESEARCH	DALLAS	12-SEP-2005
30	ALLEN	SALES	CHICAGO	12-SEP-2005
30	WARD	SALES	CHICAGO	12-SEP-2005
20	JONES	RESEARCH	DALLAS	12-SEP-2005
30	MARTIN	SALES	CHICAGO	12-SEP-2005
30	BLAKE	SALES	CHICAGO	12-SEP-2005
10	CLARK	ACCOUNTING	NEW YORK	12-SEP-2005
20	SCOTT	RESEARCH	DALLAS	12-SEP-2005
10	KING	ACCOUNTING	NEW YORK	12-SEP-2005
30	TURNER	SALES	CHICAGO	12-SEP-2005
20	ADAMS	RESEARCH	DALLAS	12-SEP-2005
30	JAMES	SALES	CHICAGO	12-SEP-2005
20	FORD	RESEARCH	DALLAS	12-SEP-2005
10	MILLER	ACCOUNTING	NEW YORK	12-SEP-2005

## 讨论

该解决方案的关键是使用对象的构造函数（默认情况下，构造函数的名字与对象名相同）。由于对象本身是单个标量值，因此它不会违反标量子查询规则，从下列查询中可以看到这点：

```
select e.deptno,
       e.ename,
       e.sal,
       (select generic_obj(d.dname,d.loc,sysdate-1)
        from dept d
        where e.deptno=d.deptno) multival
from emp e
```

DEPTNO	ENAME	SAL	MULTIVAL(VAL1, VAL2, VAL3)
20	SMITH	800	GENERIC_OBJ('RESEARCH', 'DALLAS', '12-SEP-2005')
30	ALLEN	1600	GENERIC_OBJ('SALES', 'CHICAGO', '12-SEP-2005')
30	WARD	1250	GENERIC_OBJ('SALES', 'CHICAGO', '12-SEP-2005')
20	JONES	2975	GENERIC_OBJ('RESEARCH', 'DALLAS', '12-SEP-2005')
30	MARTIN	1250	GENERIC_OBJ('SALES', 'CHICAGO', '12-SEP-2005')
30	BLAKE	2850	GENERIC_OBJ('SALES', 'CHICAGO', '12-SEP-2005')
10	CLARK	2450	GENERIC_OBJ('ACCOUNTING', 'NEW YORK', '12-SEP-2005')
20	SCOTT	3000	GENERIC_OBJ('RESEARCH', 'DALLAS', '12-SEP-2005')
10	KING	5000	GENERIC_OBJ('ACCOUNTING', 'NEW YORK', '12-SEP-2005')
30	TURNER	1500	GENERIC_OBJ('SALES', 'CHICAGO', '12-SEP-2005')
20	ADAMS	1100	GENERIC_OBJ('RESEARCH', 'DALLAS', '12-SEP-2005')
30	JAMES	950	GENERIC_OBJ('SALES', 'CHICAGO', '12-SEP-2005')
20	FORD	3000	GENERIC_OBJ('RESEARCH', 'DALLAS', '12-SEP-2005')
10	MILLER	1300	GENERIC_OBJ('ACCOUNTING', 'NEW YORK', '12-SEP-2005')

下一步，只要把查询包入内联视图中，并提取其中的各属性。

**警告：**有一点需要特别注意：Oracle 不同于其他销售商，它通常不需要为内联视图命名。然而，在这个特殊例子中，必须得给内联视图命名。否则，就不能引用对象的属性。

## 14.11 把连续数据分解为行

### 问题

有一组连续数据（存储在字符串中），要分解它们，并按行返回。例如，存储了下列数据：

```
STRINGS
-----
entry:stewiegriffin:lois:brian:
entry:moe::sizlack:
entry:petergriffin:meg:chris:
entry:willie:
entry:quagmire:mayorwest:cleveland:
entry:::flanders:
entry:robo:tchi:ken:
```

把这些连续数据转换为下列结果集：

VAL1	VAL2	VAL3
moe		sizlack
petergriffin	meg	chris
quagmire	mayorwest	cleveland
robo	tchi	ken
stewiegriffin	lois	brian
willie		
		flanders

## 解决方案

这个例子中的每个连续字符串最多包含 3 个值。这些值用冒号分隔开，而且字符串可能包含 3 项，也可能少于 3 项。如果字符串中包含的少于 3 项，则必须小心，要将存在的项放置于结果集的正确列中。例如，下面的行：

```
entry:::flanders:
```

这一行表示前两项缺失，只存在第三项。因此，如果检验“问题”部分的目标结果集，就会发现，“flanders”所在的行中 VAL1 和 VAL2 的值都为 NULL。

这个解决方案的关键是：采用字符串分解方法处理整个字符串，后面紧跟一个简单转置。该解决方案使用了视图 V 返回的行，其定义如下。这个例子采用了 Oracle 语法，但本节只需要字符串分解函数，因此转换到其他平台也是相当容易的：

```
create view V
as
select 'entry:stewiegriffin:lois:brian:' strings
  from dual
 union all
select 'entry:moe::sizlack:'
  from dual
 union all
select 'entry:petergriffin:meg:chris:'
  from dual
 union all
select 'entry:willie:'
  from dual
 union all
select 'entry:quagmire:mayorwest:cleveland:'
  from dual
 union all
select 'entry:::flanders:'
  from dual
 union all
select 'entry:robo:tchi:ken:'
  from dual
```

采用视图 V 提供要进行分解的数据，其解决方案如下所示：

```
1 with cartesian as (
2   select level id
3     from dual
4    connect by level <= 100
5 )
6 select max(decode(id,1,substr(strings,p1+1,p2-1))) val1,
7        max(decode(id,2,substr(strings,p1+1,p2-1))) val2,
8        max(decode(id,3,substr(strings,p1+1,p2-1))) val3
9   from (
10    select v.strings,
11           c.id,
12           instr(v.strings,':',1,c.id) p1,
13           instr(v.strings,':',1,c.id+1)-instr(v.strings,':',1,c.id) p2
14    from v, cartesian c
15   where c.id <= (length(v.strings)-length(replace(v.strings,':')))-1
16   )
17  group by strings
18  order by 1
```

## 讨论

第一步，遍历连续字符串：

```
with cartesian as (
  select level id
    from dual
   connect by level <= 100
)
select v.strings,
       c.id
  from v, cartesian c
 where c.id <= (length(v.strings)-length(replace(v.strings,':')))-1
```

STRINGS	ID
entry::flanders:	1
entry::flanders:	2
entry::flanders:	3
entry:moe::sizlack:	1
entry:moe::sizlack:	2
entry:moe::sizlack:	3
entry:petergriffin:meg:chris:	1
entry:petergriffin:meg:chris:	3
entry:petergriffin:meg:chris:	2
entry:quagmire:mayorwest:cleveland:	1
entry:quagmire:mayorwest:cleveland:	3
entry:quagmire:mayorwest:cleveland:	2
entry:robo:tchi:ken:	1
entry:robo:tchi:ken:	2
entry:robo:tchi:ken:	3
entry:stewiegriffin:lois:brian:	1
entry:stewiegriffin:lois:brian:	3
entry:stewiegriffin:lois:brian:	2
entry:willie:	1

下一步，使用函数 INSTR，找到每个字符串中每个冒号的位置。由于需要提取的每个值都是用两个冒号括起来的，所以“第一个”和“第二个”冒号的位置分别命名为 P1 和 P2：

```
with cartesian as (
  select level id
    from dual
   connect by level <= 100
)
select v.strings,
       c.id,
       instr(v.strings,':',1,c.id) p1,
       instr(v.strings,':',1,c.id+1)-instr(v.strings,':',1,c.id) p2
  from v, cartesian c
 where c.id <= (length(v.strings)-length(replace(v.strings,':')))-1
 order by 1
```

STRINGS	ID	P1	P2
entry::flanders:	1	6	1
entry::flanders:	2	7	1
entry::flanders:	3	8	9
entry:moe::sizlack:	1	6	4
entry:moe::sizlack:	2	10	1
entry:moe::sizlack:	3	11	8
entry:petergriffin:meg:chris:	1	6	13
entry:petergriffin:meg:chris:	3	23	6
entry:petergriffin:meg:chris:	2	19	4
entry:quagmire:mayorwest:cleveland:	1	6	9

entry:quagmire:mayorwest:cleveland:	3	25	10
entry:quagmire:mayorwest:cleveland:	2	15	10
entry:robo:tchi:ken:	1	6	5
entry:robo:tchi:ken:	2	11	5
entry:robo:tchi:ken:	3	16	4
entry:stewiegriffin:lois:brian:	1	6	14
entry:stewiegriffin:lois:brian:	3	25	6
entry:stewiegriffin:lois:brian:	2	20	5
entry:willie:	1	6	7

现在，知道了每个字符串中每对冒号的位置，只要把这个信息传递给函数 SUBSTR，就可以提取出各个值。由于要创建一个包含 3 列的结果集，需使用 DECODE 判断笛卡儿积产生的 ID：

```
with cartesian as (
select level id
  from dual
 connect by level <= 100
)
select decode(id,1,substr(strings,p1+1,p2-1)) val1,
       decode(id,2,substr(strings,p1+1,p2-1)) val2,
       decode(id,3,substr(strings,p1+1,p2-1)) val3
  from (
select v.strings,
       c.id,
       instr(v.strings,':',1,c.id) p1,
       instr(v.strings,':',1,c.id+1)-instr(v.strings,':',1,c.id) p2
  from v,cartesian c
 where c.id <= (length(v.strings)-length(replace(v.strings,':')))-1
  )
 order by 1
```

VAL1	VAL2	VAL3
moe		
petergriffin		
quagmire		
robo		
stewiegriffin		
willie		
	lois	
	meg	
	mayorwest	
	tchi	
		brian
		sizlack
		chris
		cleveland
		flanders
		ken

最后一步，将聚集函数应用于 SUBSTR 返回的值，并按 ID 分组，以便生成可读的结果集：

```
with cartesian as (
select level id
  from dual
 connect by level <= 100
)
select max(decode(id,1,substr(strings,p1+1,p2-1))) val1,
       max(decode(id,2,substr(strings,p1+1,p2-1))) val2,
       max(decode(id,3,substr(strings,p1+1,p2-1))) val3
  from (
select v.strings,
       c.id,

```

```

instr(v.strings,':',1,c.id) p1,
instr(v.strings,':',1,c.id+1)-instr(v.strings,':',1,c.id) p2
from v, cartesian c
where c.id <= (length(v.strings)-length(replace(v.strings,':')))-1
)
group by strings
order by 1

```

VAL1	VAL2	VAL3
moe		sizlack
petergriffin	meg	chris
quagmire	mayorwest	cleveland
robo	tchi	ken
stewiegriffin	lois	brian
willie		flanders

## 14.12 计算相对于总数的百分数

### 问题

要报告一组数字值，而且以占总数百分比的方式显示每个值。例如，在 Oracle 系统中，返回如下结果集：它按 JOB 显示工资细目，以便确定公司内哪个 JOB 职位花费最多。结果中也要包含每个 JOB 的员工数，以防止其结果产生误导。要生成的报表如下：

JOB	NUM_EMPS	PCT_OF_ALL_SALARIES
CLERK	4	14
ANALYST	2	20
MANAGER	3	28
SALESMAN	4	19
PRESIDENT	1	17

可以看到，如果报表中没有包含员工数，那么总裁职位拿的工资好像只占总工资的一小部分。实际上，只有一个总裁，它拿了工资总额的 17%。

### 解决方案

只有 Oracle 提供了适合该问题的解决方案，其中用到内置函数 RATIO\_TO\_REPORT。对于其他数据库，要计算占总数的百分数，可以使用除法，请参阅第 7 章的 7.10 节。

```

1 select job,num_emps,sum(round(pct)) pct_of_all_salaries
2   from (
3 select job,
4        count(*)over(partition by job) num_emps,
5        ratio_to_report(sal)over()*100 pct
6   from emp
7  )
8  group by job,num_emps

```

### 讨论

第一步，使用窗口函数 COUNT OVER，返回每个 JOB 的员工数。然后使用 RATIO\_TO\_REPORT，返回每个员工的工资占总工资的百分数（以十进制格式返回值）：

```
select job,
```



```

count(*)over(partition by job) num_emps,
ratio_to_report(sal)over()*100 pct
from emp

```

JOB	NUM_EMPS	PCT
ANALYST	2	10.3359173
ANALYST	2	10.3359173
CLERK	4	2.75624462
CLERK	4	3.78983635
CLERK	4	4.4788975
CLERK	4	3.27304048
MANAGER	3	10.2497847
MANAGER	3	8.44099914
MANAGER	3	9.81912145
PRESIDENT	1	17.2265289
SALESMAN	4	5.51248923
SALESMAN	4	4.30663221
SALESMAN	4	5.16795866
SALESMAN	4	4.30663221

最后一步，使用聚集函数 SUM，计算 RATIO\_TO\_REPORT 返回值的总和。确保按 JOB 和 NUM\_EMPS 分组。乘 100，返回一个表示百分数的整数（例如，25% 返回 25 而不是 0.25）：

```

select job,num_emps,sum(round(pct)) pct_of_all_salaries
  from (
select job,
count(*)over(partition by job) num_emps,
ratio_to_report(sal)over()*100 pct
  from emp
)
group by job,num_emps

```

JOB	NUM_EMPS	PCT_OF_ALL_SALARIES
CLERK	4	14
ANALYST	2	20
MANAGER	3	28
SALESMAN	4	19
PRESIDENT	1	17

## 14.13 从 Oracle 创建 CSV 格式输出

### 问题

用某个表中的行创建一个分隔列表（可能用逗号分隔）。例如，使用表 EMP，返回下列结果集：

```

DEPTNO LIST
-----
10 MILLER,KING,CLARK
20 FORD,ADAMS,SCOTT,JONES,SMITH
30 JAMES,TURNER,BLAKE,MARTIN,WARD,ALLEN

```

在 Oracle 系统（Oracle Database 10g 或更高版本）中，并且要使用 MODEL 子句。

### 解决方案

该解决方案使用了 Oracle 中 MODEL 子句的叠代功能。其技巧是使用窗口函数

ROW\_NUMBER OVER 给 DEPTNO 中的每个员工分等级（按 EMPNO，只是随意选择的）。由于 MODEL 提供了以数组方式访问行，则通过对等级进行减法操作，可以访问前面的数组元素。这样，为每个员工创建一个列表，它包含员工的姓名以及等级在当前员工之前的员工姓名：

```

1  select deptno,
2         list
3  from (
4  select *
5  from (
6  select deptno, empno, ename,
7         lag(deptno) over (partition by deptno
8                          order by empno) prior_deptno
9  from emp
10 )
11 model
12   dimension by
13   (
14     deptno,
15     row_number() over (partition by deptno order by empno) rn
16   )
17   measures
18   (
19     ename,
20     prior_deptno, cast(null as varchar2(60)) list,
21     count(*) over (partition by deptno) cnt,
22     row_number() over (partition by deptno order by empno) rnk
23   )
24   rules
25   (
26     list[any, any]
27     order by deptno, rn = case when prior_deptno[cv(), cv()] is null
28                               then ename[cv(), cv()]
29                               else ename[cv(), cv()] || ', ' ||
30                                     list[cv(), rnk[cv(), cv()]-1]
31                               end
32   )
33 )
34 where cnt = rn

```

## 讨论

第一步，使用窗口函数 LAG OVER，返回前一个员工（按 EMPNO 顺序）的 DEPTNO。结果是按 DEPTNO 分区的，所以部门中第一个员工（按 EMPNO 顺序）的返回值为 NULL，而其余员工的返回值为 DEPTNO。结果如下所示：

```

select deptno, empno, ename,
       lag(deptno) over (partition by deptno
                        order by empno) prior_deptno
from emp

```

DEPTNO	EMPNO	ENAME	PRIOR_DEPTNO
10	7782	CLARK	
10	7839	KING	10
10	7934	MILLER	10
20	7369	SMITH	
20	7566	JONES	20
20	7788	SCOTT	20
20	7876	ADAMS	20

20	7902 FORD	20
30	7499 ALLEN	
30	7521 WARD	30
30	7654 MARTIN	30
30	7698 BLAKE	30
30	7844 TURNER	30
30	7900 JAMES	30

下一步，检验 MODEL 子句的 MEASURES 小子句。MEASURES 列表中的项就是数组：

*ENAME*

这是 EMP 中所有 ENAME 组成的数组

*PRIOR\_DEPTNO*

这是由 LAG OVER 窗口函数返回的值组成的数组

*CNT*

这是每个 DEPTNO 中员工数组成的数组

*RNK*

这是每个 DEPTNO 中所有员工等级（按 EMPNO）的数组

数组下标是 DEPTNO 和 RN（DIMENSION BY 小子句中的 ROW\_NUMBER OVER 窗口函数返回的值）。要查看这些数组包含的内容，只要注释掉 MODEL 子句中 RULES 小子句列出的代码，然后执行查询即可，如下所示：

```
select *
  from (
select deptno,empno,ename,
       lag(deptno)over(partition by deptno
                       order by empno) prior_deptno
  from emp
  )
model
  dimension by
  (
    deptno,
    row_number()over(partition by deptno order by empno) rn
  )
  measures
  (
    ename,
    prior_deptno,cast(null as varchar2(60)) list,
    count(*)over(partition by deptno) cnt,
    row_number()over(partition by deptno order by empno) rnk
  )
  rules
  (
/*
    list[any,any]
    order by deptno,rn = case when prior_deptno[cv(),cv()] is null
                           then ename[cv(),cv()]
                           else ename[cv(),cv()]||', '||
                               list[cv(),rnk[cv(),cv()]]-1]
    end
*/
  )
order by 1
```

DEPTNO	RN	ENAME	PRIOR_DEPTNO	LIST	CNT	RNK
10	1	CLARK			3	1
10	2	KING	10		3	2
10	3	MILLER	10		3	3
20	1	SMITH			5	1
20	2	JONES	20		5	2
20	4	ADAMS	20		5	4
20	5	FORD	20		5	5
20	3	SCOTT	20		5	3
30	1	ALLEN			6	1
30	6	JAMES	30		6	6
30	4	BLAKE	30		6	4
30	3	MARTIN	30		6	3
30	5	TURNER	30		6	5
30	2	WARD	30		6	2

现在已经知道了 MODEL 子句中所声明的每一项的作用，下面继续探讨 RULES 小子句。如果查看 CASE 表达式，会看到已经计算了 PRIOR\_DEPTNO 的当前值。如果这个值为 NULL，它表示应该把每个 DEPTNO 和 ENAME 中的第一个员工返回到当前员工的 LIST 数组中；如果 PRIOR\_DEPTNO 的值不是 NULL，那么把前一个员工的 LIST 值加到当前员工的姓名（ENAME 数组）中，然后，把这个结果作为当前员工的 LIST 返回。在处理 DEPTNO 中的每一行时，该 CASE 表达式运算的结果就是用迭代的方式建立一个用逗号分隔的（CSV 格式）值列表。在下面的例子中，可以看到中间结果：

```

select deptno,
       list
from (
select *
from (
select deptno, empno, ename,
       lag(deptno) over (partition by deptno
                        order by empno) prior_deptno
from emp
)
model
dimension by
(
deptno,
row_number() over (partition by deptno order by empno) rn
)
measures
(
ename,
prior_deptno, cast(null as varchar2(60)) list,
count(*) over (partition by deptno) cnt,
row_number() over (partition by deptno order by empno) rnk
)
rules
(
list[any,any]
order by deptno, rn = case when prior_deptno[cv(),cv()] is null
                        then ename[cv(),cv()]
                        else ename[cv(),cv()]||', '||
                        list[cv(),rnk[cv(),cv()]-1]
                        end
)
)
DEPTNO LIST
-----
10 CLARK

```

```

10 KING, CLARK
10 MILLER, KING, CLARK
20 SMITH
20 JONES, SMITH
20 SCOTT, JONES, SMITH
20 ADAMS, SCOTT, JONES, SMITH
20 FORD, ADAMS, SCOTT, JONES, SMITH
30 ALLEN
30 WARD, ALLEN
30 MARTIN, WARD, ALLEN
30 BLAKE, MARTIN, WARD, ALLEN
30 TURNER, BLAKE, MARTIN, WARD, ALLEN
30 JAMES, TURNER, BLAKE, MARTIN, WARD, ALLEN

```

最后一步，保留每个 DEPTNO 中的最后一名员工，以确保每个 DEPTNO 都包含完整的 CSV 列表。使用 CNT 数组存储的值及 RN 数组存储的值，就可以为每个 DEPTNO 保留完整的 CSV。因为 RN 表示每个 DEPTNO 中员工（按 EMPNO）的等级，所以每个 DEPTNO 中的最后一名员工将满足条件  $CNT = RN$ ，如下例所示：

```

select deptno,
       list
  from (
select *
  from (
select deptno, empno, ename,
       lag(deptno) over (partition by deptno
                        order by empno) prior_deptno
  from emp
  )
 model
  dimension by
    (
      deptno,
      row_number() over (partition by deptno order by empno) rn
    )
  measures
    (
      ename,
      prior_deptno, cast(null as varchar2(60)) list,
      count(*) over (partition by deptno) cnt,
      row_number() over (partition by deptno order by empno) rnk
    )
  rules
    (
      list[any, any]
      order by deptno, rn = case when prior_deptno[cv(), cv()] is null
                                then ename[cv(), cv()]
                                else ename[cv(), cv()] || ',' ||
                                list[cv(), rnk[cv(), cv()]-1]
                                end
    )
  )
 where cnt = rn
DEPTNO LIST
-----
10 MILLER, KING, CLARK
20 FORD, ADAMS, SCOTT, JONES, SMITH
30 JAMES, TURNER, BLAKE, MARTIN, WARD, ALLEN

```

## 14.14 找到与模式不匹配的文本 (Oracle)

### 问题

有一个文本字段，它包含一些结构化的文本值（例如，电话号码），要找到其中结构不正确的值出现的次数。例如，有下列数据：

```
select emp_id, text
  from employee_comment
```

EMP_ID	TEXT
7369	126 Varnum, Edmore MI 48829, 989 313-5351
7499	1105 McConnell Court Cedar Lake MI 48812 Home: 989-387-4321 Cell: (237) 438-3333

希望列出电话号码格式不正确的行。例如，列出下面这行，原因是它的电话号码使用了两个不同的分隔字符：

7369	126 Varnum, Edmore MI 48829, 989 313-5351
------	---

电话号码中的两个分隔字符相同才认为是格式正确。

### 解决方案

这个问题的解决方案包含下列步骤：

1. 找到一种方式，描述电话号码的外观格式。
2. 将格式正确的电话号码排除。
3. 查看是否还有一些其他外观电话号码遗留下来，如果有，则说明它们的格式不正确。

下列解决方案充分利用了 Oracle Database 10g 引入的正则表达式功能。

```
select emp_id, text
  from employee_comment
 where regexp_like(text, '[0-9]{3}[-. ] [0-9]{3}[-. ] [0-9]{4}')
    and regexp_like(
        regexp_replace(text,
            '[0-9]{3}([-. ] [0-9]{3})\1[0-9]{4}', '****'),
        '[0-9]{3}[-. ] [0-9]{3}[-. ] [0-9]{4}')
```

EMP_ID	TEXT
7369	126 Varnum, Edmore MI 48829, 989 313-5351
7844	989-387.5359
9999	906-387-1698, 313-535.8886

在这些行中，每行至少包含一个格式不正确的电话号码。

### 讨论

这个解决方案的关键是：检测“电话号码外观”。假定电话号码都存储在注释字段中，那

么该字段中的文本都可以构造成无效的电话号码。采用一种方式，把字段缩小到更合理的信息组。例如，在输出中不会看到下列行：

```
EMP_ID TEXT
```

```
-----  
7900 Cares for 100-year-old aunt during the day. Schedule only  
      for evening and night shifts.
```

很显然，这一行根本不包含电话号码，更谈不上什么格式错误。问题是如何让 RDBMS “看出”这一点呢？读者一定很想知道答案，那么继续往下读吧。

---

注意：本节的内容取自（已获得许可）Jonathan Gennick 写的一篇题为“Regular Expression Anti-Patterns”的文章。访问网站：<http://gennick.com/antiregex.htm>，可以阅读该文章。

---

这个解决方案使用 Pattern A，定义电话号码“外观”：

```
Pattern A: [0-9]{3}[-. ] [0-9]{3}[-. ] [0-9]{4}
```

Pattern A 检查两组 3 位数字，后面跟随一组 4 位数字，组与组之间，可以采用破折号 (-)、句号 (.) 或空格作为分隔符。也可以采用更复杂的模式。例如，也希望考虑 7 位数字的电话号码，但这里不会讨论。现在，关键是要定义电话号码字符串的模式，对于这个问题，其模式是由 Pattern A 定义的。也可以定义不同的 Pattern A，而且该总体解决方案依然适用。

此解决方案在 WHERE 子句中使用了 Pattern A，以确保只把包含“可能是电话号码（由模式定义）”的行显示出来：

```
select emp_id, text  
  from employee_comment  
 where regexp_like(text, '[0-9]{3}[-. ] [0-9]{3}[-. ] [0-9]{4}')
```

接下来，需要定义“好的”电话号码看起来什么样。本解决方案使用 Pattern B 完成此功能：

```
Pattern B: [0-9]{3}([-. ]) [0-9]{3}\1[0-9]{4}
```

这次，模式使用“\1”引用第一个子表达式，与“([-. ])”匹配的字符一定要与“\1”匹配。Pattern B 描述了“好”的电话号码，需要将它们排除掉，不予考虑（因为它们不是“坏的”）。该解决方案调用 REGEXP\_REPLACE 排除格式完美的电话号码：

```
regexp_replace(text,  
  '[0-9]{3}([-. ]) [0-9]{3}\1[0-9]{4}', '****'),
```

对 REGEXP\_REPLACE 的调用在 WHERE 子句中进行，用三个星号的字符串代替格式正确的电话号码。同样，Pattern B 可以是其他任何希望的模式，其要点是 Pattern B 描述了可接受的模式。

用三个星号的字符串 (\*\*\*) 替换格式正确的电话号码之后, 根据定义, 保留下来的“外观”电话号码格式一定有问题。解决方案把 REGEXP\_LIKE 应用于 REGEXP\_REPLACE (这里原文是 REGEXP\_LIKE, 有问题, 译者注) 的输出, 以查看是否有格式错误的电话号码留下来:

```
and regexp_like(
    regexp_replace(text,
        '[0-9]{3}([-.]?[0-9]{3})\1[0-9]{4}', '****'),
    '[0-9]{3}([-.]?[0-9]{3}([-.]?[0-9]{4})')
```

如果没有模式匹配功能 (这是 Oracle 相对较新的正则表达式的固有特性), 本节的问题会很难解决。特别地, 本节依赖于 REGEXP\_REPLACE, 其他数据库 (著名的有 PostgreSQL) 也支持正则表达式, 但是, 据我所知, 只有 Oracle 支持正则表达式的搜索和替换功能, 本节就依赖于这些功能。

## 14.15 用内联视图转换数据

### 问题

有一个表, 其中的一列有时包含数字数据, 有时包含字符数据, 而同一个表中的另外一列指明了是字符还是数字。现希望使用子查询只分离出数字数据:

```
select *
from ( select flag, to_number(num) num
      from subtest
      where flag in ('A', 'C') )
where num > 0
```

令人遗憾的是, 这个针对内联视图的查询经常 (但不总是!) 会产生下列错误消息:

```
ERROR:
ORA-01722: invalid number
```

### 解决方案

一种解决方案是强制内联视图在外部 SELECT 语句之前执行完毕。至少在 Oracle 中, 通过在内层 SELECT 列表中包含行序号伪列, 可以做到这一点:

```
select *
from ( select rownum, flag, to_number(num) num
      from subtest
      where flag in ('A', 'C') )
where num > 0
```

对于该解决方案起作用的原因, 请参阅“讨论”。

### 讨论

在有问题查询中产生无效数字错误的原因是: 有些优化程序会将内层查询和外层查询合并起来, 尽管看起来好像是先执行内层查询, 以排除所有非数字 NUM 值, 但真正执行的很可能是:



```
select flag, to_number(num) num
from subtest
where to_number(num) > 0 and flag in ('A', 'C');
```

现在，可以清楚地看到出错的原因：在执行 TO\_NUMBER 函数之前，并没有筛选出包含非数字 NUM 值的行。

---

注意：数据库应该把子查询和主查询合并起来吗？答案取决于考虑的角度是关系理论、SQL 标准还是特定数据库厂家实现 SQL 时所做的选择。要了解更多内容，请访问 <http://gennick.com/madness.html> 网站。

---

这个解决方案至少解决了 Oracle 中问题，其原因是它在内层查询的 SELECT 列表中加入了 ROWNUM。ROWNUM 是一个函数，它为查询返回的每一行返回连续递增的序号。最后这几个词非常重要，连续递增的序号，也即行号，不能从查询返回行的上下文之外计算出来。这样，强制 Oracle 实际产生子查询的结果，也就意味着强制 Oracle 为了正确地分配行号，首先必须从该子查询中得到返回行，而要得到返回行，必须先执行子查询。因此，查询 ROWNUM 是一种机制，可以用它强制 Oracle 在执行主查询之前完全执行子查询（即不允许合并查询）。如果没有使用 Oracle，而且需要强制首先执行子查询，则检查所用数据库是否支持类似 Oracle 的 ROWNUM 函数。

## 14.16 测试一个组内是否存在某个值 问题

依据组内是否有某行包含特定值，为组内各行创建一个布尔标志。考虑下面的例子：一个学生在一段时间内参加了几次考试。某个学生三个月内参加了三次考试，如果通过了其中一次考试，就满足了要求，此时应该返回一个标志，表示这个事实。如果他在三个月内一次也没有通过，则返回另一个标志，也表示该事实。请看下面的例子（使用 Oracle 语法组成这个例子的行；对于 DB2 和 SQL Server，需要进行一点修改，因为二者都支持窗口函数）：

```
create view V
as
select 1 student_id,
       1 test_id,
       2 grade_id,
       1 period_id,
       to_date('02/01/2005','MM/DD/YYYY') test_date,
       0 pass_fail
from dual union all
select 1, 2, 2, 1, to_date('03/01/2005','MM/DD/YYYY'), 1 from dual union all
select 1, 3, 2, 1, to_date('04/01/2005','MM/DD/YYYY'), 0 from dual union all
select 1, 4, 2, 2, to_date('05/01/2005','MM/DD/YYYY'), 0 from dual union all
select 1, 5, 2, 2, to_date('06/01/2005','MM/DD/YYYY'), 0 from dual union all
select 1, 6, 2, 2, to_date('07/01/2005','MM/DD/YYYY'), 0 from dual

select *
from V
```

STUDENT_ID	TEST_ID	GRADE_ID	PERIOD_ID	TEST_DATE	PASS_FAIL
1	1	2	1	01-FEB-2005	0
1	2	2	1	01-MAR-2005	1
1	3	2	1	01-APR-2005	0
1	4	2	2	01-MAY-2005	0
1	5	2	2	01-JUN-2005	0
1	6	2	2	01-JUL-2005	0

检验上面的结果集可以看到，该学生在两个为期三个月的期间内参加了 6 次考试，他通过了一次考试（1 意味着“通过”，0 意味着“未通过”），那么整个第一个期间就满足了要求；他在第二个期间（后三个月）没有通过任何考试，这三次考试的 PASS\_FAIL 都为 0。返回一个结果集，它展示学生是否在给定期限内通过考试。最后，返回以下结果集：

STUDENT_ID	TEST_ID	GRADE_ID	PERIOD_ID	TEST_DATE	METREQ	IN_PROGRESS
1	1	2	1	01-FEB-2005	+	0
1	2	2	1	01-MAR-2005	+	0
1	3	2	1	01-APR-2005	+	0
1	4	2	2	01-MAY-2005	-	0
1	5	2	2	01-JUN-2005	-	0
1	6	2	2	01-JUL-2005	-	1

METREQ 的值（“满足要求”）是“+”和“-”，分别表示学生已经满足或未满足要求，即在归定期间（跨度三个月）至少通过一次考试。如果学生在规定期间通过了考试，则 IN\_PROGRESS 的值应该是 0；如果学生在规定期间没有通过考试，那么这个学生最后考试日期对应行的 IN\_PROGRESS 值应为 1。

## 解决方案

使这个问题有点棘手的原因是把同组的行当作组而不是个体来处理。想一想“问题”部分 PASS\_FAIL 的值，如果一行接一行的计算，除了 TEST\_ID 2 之外，似乎每行的 METREQ 值都应该是“-”，但事实并非如此。必须确保把行当作组进行处理。通过使用窗口函数 MAX OVER，很容易就可以确定学生是否在一个特殊时期至少通过了一次考试。一旦有了这个信息，得到“布尔”值就是非常简单的事情，只需使用 CASE 表达式：

```

1  select student_id,
2         test_id,
3         grade_id,
4         period_id,
5         test_date,
6         decode( grp_p_f,1,lpad('+',6),lpad('-',6) ) metreq,
7         decode( grp_p_f,1,0,
8                decode( test_date,last_test,1,0 ) ) in_progress
9  from (
10 select V.*,
11        max(pass_fail)over(partition by
12                           student_id,grade_id,period_id) grp_p_f,
13        max(test_date)over(partition by
14                           student_id,grade_id,period_id) last_test
15  from V
16  ) x

```

## 讨论

解决方案的关键是：使用窗口函数 MAX OVER，返回每个组中 PASS\_FAIL 的最大值。由于 PASS\_FAIL 的值只有 1 或 0，因此如果学生至少通过了一次考试，MAX OVER 就会为整个组返回 1。其操作过程如下所示：

```
select V.*,
       max(pass_fail)over(partition by
                           student_id,grade_id,period_id) grp_pass_fail
from V
```

STUDENT_ID	TEST_ID	GRADE_ID	PERIOD_ID	TEST_DATE	PASS_FAIL	GRP_PASS_FAIL
1	1	2	1	01-FEB-2005	0	1
1	2	2	1	01-MAR-2005	1	1
1	3	2	1	01-APR-2005	0	1
1	4	2	2	01-MAY-2005	0	0
1	5	2	2	01-JUN-2005	0	0
1	6	2	2	01-JUL-2005	0	0

上面的结果集显示了学生在第一个期间至少通过了一次考试，这样，整个组的值都为1或“通过”。下一个要求就是如果学生在一个期间没有通过任何考试，则会为那个组中最新考试日期的 IN\_PROGRESS 标志返回 1 值。也可以使用窗口函数 MAX OVER 完成这件事：

```
select V.*,
       max(pass_fail)over(partition by
                           student_id,grade_id,period_id) grp_p_f,
       max(test_date)over(partition by
                           student_id,grade_id,period_id) last_test
from V
```

STUDENT_ID	TEST_ID	GRADE_ID	PERIOD_ID	TEST_DATE	PASS_FAIL	GRP_P_F	LAST_TEST
1	1	2	1	01-FEB-2005	0	1	01-APR-2005
1	2	2	1	01-MAR-2005	1	1	01-APR-2005
1	3	2	1	01-APR-2005	0	1	01-APR-2005
1	4	2	2	01-MAY-2005	0	0	01-JUL-2005
1	5	2	2	01-JUN-2005	0	0	01-JUL-2005
1	6	2	2	01-JUL-2005	0	0	01-JUL-2005

现在，已经确定了学生哪个期间通过了考试以及每个期间最后一次考试日期是什么。那么最后一步就是进行格式设置，使结果集看起来更优美。最终解决方案使用 Oracle 的 DECODE 函数，以创建 METREQ 和 IN\_PROGRESS 列。使用 LPAD 函数，将 METREQ 的值右对齐：

```
select student_id,
       test_id,
       grade_id,
       period_id,
       test_date,
       decode( grp_p_f,1,lpad('+',6),lpad('-',6) ) metreq,
       decode( grp_p_f,1,0,
               decode( test_date,last_test,1,0 ) ) in_progress
from (
select V.*,
       max(pass_fail)over(partition by
                           student_id,grade_id,period_id) grp_p_f,
       max(test_date)over(partition by
```

```

                                student_id,grade_id,period_id) last_test
from V
) x

```

STUDENT_ID	TEST_ID	GRADE_ID	PERIOD_ID	TEST_DATE	METREQ	IN_PROGRESS
1	1	2	1	01-FEB-2005	+	0
1	2	2	1	01-MAR-2005	+	0
1	3	2	1	01-APR-2005	+	0
1	4	2	2	01-MAY-2005	-	0
1	5	2	2	01-JUN-2005	-	0
1	6	2	2	01-JUL-2005	-	1

## 窗口函数补充

在本书中，充分发挥了 2003 年新加入 ISO SQL 标准的窗口函数，以及各数据库厂商的特定窗口函数的优势。本附录意图对窗口函数的机理做个简要回顾。窗口函数能够使很多典型的难题（也就是说，采用标准 SQL 很难解决的问题）变得相当容易。对于可用窗口函数的完整列表、全部语法以及有关工作机理的深入介绍，请查阅相关厂商的文档。

### A.1 分组

在讨论窗口函数之前，先了解 SQL 中分组的操作方式是至关重要的。根据笔者的经历，在 SQL 中给结果分组的概念已经成为很多人的一个障碍，问题的根本在于它们并不完全了解 GROUP BY 子句的工作机理以及使用 GROUP BY 时某些特定的查询为什么会返回某些特定的结果。

简言之，分组是把类似的行组织在一起的一种方式。当在查询中使用 GROUP BY 时，结果集中的每一行都是一个组，而且表示一行（或多行）中的某一列（或多列）具有相同值。这就是其要旨所在。

如果某个组只是一行的唯一实例，该行表示一行或多行的某个特定列（或某些列）具有相同值，那么表 EMP 中有关组的实际例子就有“部门 10 中的所有员工”（同一组中的所有员工，都有一个共同值，即 DEPTNO=10）或“所有职员”（同一组中的职员共同值是 JOB='CLERK'）。请看下面的查询。首先显示了部门 10 中的所有员工；第二个查询给部门 10 中的员工分组，而且返回有关组的下列信息：组中的行（成员）数、最高工资和最低工资：

```
select deptno,ename
  from emp
 where deptno=10
```

```
DEPTNO ENAME
-----
    10 CLARK
    10 KING
```

```

10 MILLER
select deptno,
       count(*) as cnt,
       max(sal) as hi_sal,
       min(sal) as lo_sal
  from emp
 where deptno=10
 group by deptno

```

DEPTNO	CNT	HI_SAL	LO_SAL
10	3	5000	1300

如果不能给部门10中的员工分组，以获得上述第二个查询中的信息，那么不得不手动查看该部门包含的行（如果只有三行，就很简单，但如果有一百万行，怎么办呢？）。为什么有人想进行分组呢？这样做的理由各不相同，也许要查看存在多少个不同的组，或想看看每个组中有多少个成员（行）。从上面的简单例子可以看出，分组就能获得一个表中有关很多行的信息，而无需一个个地查看它们。

## SQL 组的定义

在数学中，通常把组定义为  $(G, \cdot, e)$ ，其中  $G$  是聚集， $\cdot$  是  $G$  上的二目运算， $e$  是  $G$  的一个成员，这里把该定义当作 SQL 组的基础。把 SQL 组定义为  $(G, e)$ ，其中  $G$  是使用 GROUP BY 的单个查询或自包含查询的结果集， $e$  是  $G$  的一个成员，而且必须满足下列规则：

- 对于  $G$  中的每个  $e$  都是独特（各不相同）的，而且表示  $e$  的一个或多个实例。
- 对于  $G$  中的每个  $e$ ，聚集函数 COUNT 会返回一个大于 0 的值。

---

**注意：**在 SQL 组的定义中引入了结果集，目的是强调在使用查询时什么是组。这样，在上述规则中用“行”替换“ $e$ ”将非常准确，因为从技术上讲，结果集中的行就代表了组。

---

由于这些性质是理解组的基础，所以证明它们的正确性就很重要（接下去会通过 SQL 查询的一些例子加以证明）。

### 组是非空的

根据它的定义，组至少包含一个成员（或行）。如果接受了这个事实，那么就说明不能从空表创建组。为证明这种说法的正确性，可简单地采用反证法。下面的例子创建了一个空表，那么试图采用三个不同的查询针对这个空表创建组：

```

create table fruits (name varchar(10))

select name
  from fruits
 group by name

(no rows selected)

```

```
select count(*) as cnt
  from fruits
 group by name

(no rows selected)

select name, count(*) as cnt
  from fruits
 group by name

(no rows selected)
```

从这些查询中可以看到，从空表创建 SQL 定义的组是不可能的。

### 组是独特的

下面就证明使用带有 GROUP BY 子句的查询创建的组是独特的。下面的例子给 FRUITS 表插入了 5 行，然后从这些行创建组：

```
insert into fruits values ('Oranges')
insert into fruits values ('Oranges')
insert into fruits values ('Oranges')
insert into fruits values ('Apple')
insert into fruits values ('Peach')

select *
  from fruits

NAME
-----
Oranges
Oranges
Oranges
Apple
Peach

select name
  from fruits
 group by name

NAME
-----
Apple
Oranges
Peach

select name, count(*) as cnt
  from fruits
 group by name

NAME          CNT
-----
Apple         1
Oranges       3
Peach         1
```

第一个查询显示了表 FRUITS 中 “Oranges” 出现了三次。然而，第二个查询和第三个查询（使用 GROUP BY）将仅返回 “Oranges” 的一个实例。这些查询就证明了结果集中的行（也即 G 中的 e）是独特的，NAME 的每个值表示表 FRUITS 中它自身的一个或多个实例。



知道组是独特的，这相当重要，也就意味着：当查询中使用 GROUP BY 时，SELECT 列表中就不必使用 DISTINCT 关键字。

注意：不能说 GROUP BY 和 DISTINCT 相同。它们表示两个完全不同的概念。结果集中 GROUP BY 子句列出的项是独特的，同时使用 DISTINCT 和 GROUP BY 是多余的。

### COUNT 绝不为 0

上一节的查询和结果也证明了最后一个定理，即当针对非空表的查询（包含 GROUP BY）中使用聚集函数 COUNT 时，它绝不会返回 0。不可能从一个组返回为 0 计数值，对此不应该感到惊讶。前面已经证明了不能从空表中创建组，因此，组至少包含一行；既然至少有一行存在，那么返回值至少为 1。

注意：记住，现在讨论的是使用了包含 GROUP BY 的 COUNT，而不是使用 COUNT 自身。对空表使用 COUNT（不包含 GROUP BY）的查询，当然可能返回 0。

## Frege 定理和 Russell 悖论

致有兴趣者。Frege 的抽象定理是以 Cantor 为无限的或不可数的集定义集成员的解决方案为基础的，该定理表明：对于一个特定的识别属性，一定存在一个集合，其成员都有此属性。Robert Stoll 提出，“麻烦的根源在于对抽象原理的滥用。”Bertrand Russell 请 Gottlob Frege 仔细考虑如下集合，它的成员也是集合，而且具有它们自身的成员所不具有的定义属性。

Russell 指出，抽象定理提供了太大的自由度，因为只要指定一个条件或属性就可以 = 来定义集合成员，这样就会产生矛盾。为更好地解释如何才能发现矛盾，他设计了“理发师难题”。理发师难题是这样的：

在一个城镇中，有一个男理发师，他给所有不自己刮胡子的男人刮胡子。如果是这样，那么谁给理发师刮胡子呢？

对于更具体的例子，请看如下定义的集合：

y 中满足特定条件 (P) 的所有成员 x

这种描述的数学符号是：

$$\{x \in y \mid P(x)\}$$

由于上面的集只考虑了 y 中满足条件 (P) 的 x，这样描述该集合可能更直观：当且仅当 x 满足条件 (P) 时，x 是 y 的成员。

—待续—



这里，把条件  $P(x)$  定义为  $x$  不是  $x$  的成员：

$$(x \in x)$$

现在该集合的定义成了：当且仅当  $x$  不是  $x$  的成员时， $x$  是  $y$  的成员：

$$\{x \in y \mid (x \in x)\}$$

读者可能并不理解 Russell 的悖论，但要问问自己：上面的集合是它自己的成员吗？现在假定  $x = y$ ，再看看上面的集合。可以把下面的集定义为：当且仅当  $y$  不是  $y$  的成员时， $y$  是  $y$  的成员：

$$\{y \in y \mid (y \in y)\}$$

简而言之，Russell 的悖论允许有一个这样的集，它是自己的成员，同时也不是自己的成员，这就是矛盾。直觉思考会使人相信这根本不是问题；那么，一个集如何成为自己的成员呢？毕竟，所有书的集并不是一本书。这个悖论为什么存在，而且它怎么是问题呢？当考虑集理论的抽象应用时，它就变成一个问题。例如，考虑一下所有集合的集合，可以说明 Russell 的悖论的“实际”应用。如果允许这样的概念存在，那么按照它的定义，它肯定是自己的成员（不管怎么说，它是所有集合的集合）。当把上面的  $P(x)$  设为所有集合的集合时，会发生什么现象呢？简言之，Russell 的悖论表明当且仅当所有集合的集合不是自己的成员时，它是自己的成员（），很显然，这是矛盾的。

令人欣喜的是，Ernst Zermelo 开发了分离定理模式（也称为子集定理模式或规范定理），在定理的集合理论中巧妙地回避 Russell 的悖论。

## 悖论

“在一个科研工作者完成自己的工作之后，却得知知识体系的基础被动摇，没有比这更不幸的了……这是在本书马上要出版印刷之际 Mr. Bertrand Russell 写给我的一封信中的一句话。”

上面这段话选自 Gottlob Frege 对 Bertrand Russell 针对 Frege 的有关集合理论的抽象定理的矛盾发现的回复。

悖论多次提供的场景似乎与已建立的理论或观点相矛盾。在很多情况下，这些矛盾带有局限性，而且可以规避，或者它们只适用于很少的测试用例，可以安全地忽略它们。

至此，读者可能会猜测：以上之所以花这么多笔墨讨论悖论，是因为在 SQL 组的定义中也存在着悖论，而且需要指出这个悖论。尽管现在集中在关于组的讨论上，实际上却是在讨论 SQL 查询。在查询的 GROUP BY 子句中，可以包含各种各样的值，例如，常量、表达式或（更常用的）表中的列。为了得到这种灵活性，是需要付出点儿代价的，这就

是在 SQL 中 NULL 是合法的“值”。由于聚集函数会忽略 NULL，所以它们会带来问题。既然如此，如果某个表只包含一行，而且它的值是 NULL，那么在 GROUP BY 查询中使用它时，聚集函数 COUNT 会返回什么呢？根据定义，当使用 GROUP BY 和聚集函数 COUNT 时，一定会返回一个大于 1 的值，那么，当函数（如 COUNT）忽略这个值时会发生什么呢？这对 GROUP 的定义来说意味着什么呢？请看下面的例子，它揭示了 NULL 在分组中的悖论（为使可读性更好，必要时使用函数 COALESCE）：

```
select *
  from fruits

NAME
-----
Oranges
Oranges
Oranges
Apple
Peach

insert into fruits values (null)
insert into fruits values (null)
insert into fruits values (null)
insert into fruits values (null)
insert into fruits values (null)

select coalesce(name,'NULL') as name
  from fruits

NAME
-----
Oranges
Oranges
Oranges
Apple
Peach
NULL
NULL
NULL
NULL
NULL

select coalesce(name,'NULL') as name,
       count(name) as cnt
  from fruits
 group by name

NAME          CNT
-----
Apple          1
NULL           0
Oranges        3
Peach          1
```

如果表中出现 NULL 值，似乎就给 SQL 组的定义带来了矛盾或悖论。幸运的是，这种矛盾并不是真正要关注的问题，在聚集函数的实现中还有比定义中多得多得有关悖论的问题需要处理。请看上述集合的最后一个查询，将这个查询用文字表述就是：

计算表 FRUITS 中每个名字出现的次数，或计算每个组中的成员数。

检验上面的 INSERT 语句，很显然，有五行包含 NULL 值，这就意味着存在一个 NULL 组，它包含 5 个成员。

注意：由于 NULL 显然具有一些区别于其他值的属性，它不仅是个值，也可能是组。

那么，当遵从组的定义时，如何编写查询，才能返回 5 而不是 0，从而返回要查找的信息呢？下面的例子显示了处理 NULL 组悖论的应对措施：

```
select coalesce(name, 'NULL') as name,
       count(*) as cnt
  from fruits
 group by name
```

NAME	CNT
Apple	1
Oranges	3
Peach	1
NULL	5

该方案使用了 COUNT(\*), 而未使用 COUNT(NAME), 这就避免了 NULL 组悖论。如果传递给聚集函数的列中存在 NULL 值，它们将忽略这些值。因此，为避免出现 0 值，使用 COUNT 时，就不要传递列名，而是传递一个星号 (\*)。星号 (\*) 能够让 COUNT 函数对行计数，而不对实际的列值计数，因此，不管实际值是否为 NULL，都无关紧要。

还有一个悖论与下面的定理有关：即结果集中的每个组 (G 中的每个 e) 都是独特的。由于 SQL 结果集和表的性质，可能可以返回一个包含重复组的结果集，因此把它们定义为多集或“袋”，而不是集合（因为允许重复行）可能更准确。请看下面的查询：

```
select coalesce(name, 'NULL') as name,
       count(*) as cnt
  from fruits
 group by name
 union all
select coalesce(name, 'NULL') as name,
       count(*) as cnt
  from fruits
 group by name
```

NAME	CNT
Apple	1
Oranges	3
Peach	1
NULL	5
Apple	1
Oranges	3
Peach	1
NULL	5

```
select x.*
  from (
select coalesce(name, 'NULL') as name,
       count(*) as cnt
  from fruits
```

```

group by name
) x,
(select deptno from dept) y

```

NAME	CNT
Apple	1
Apple	1
Apple	1
Apple	1
Oranges	3
Oranges	3
Oranges	3
Oranges	3
Peach	1
Peach	1
Peach	1
Peach	1
NULL	5
NULL	5
NULL	5
NULL	5

从这些查询中可以看到，在最终结果集中组是重复的。令人欣喜的是，不必太担心，因为它只表示局部悖论。组的第一个属性表明：对于 (G,e)，G 是结果集，它来自一个使用 GROUP BY 的单个或自包含查询。简言之，每个 GROUP BY 查询生成的结果集符合组的定义。只有当合并两个 GROUP BY 查询的结果集时，才会创建包含重复组的多集。上述例子中的查询使用了 UNION ALL，它不是一个集合操作，而是多集操作，也调用了两次 GROUP BY，从而有效执行了两次查询。

---

**注意：**如果使用 UNION（它是一个集合操作），就不会看到重复组。

---

上述集合的第二个查询使用了笛卡儿积，只有先实现分组，然后执行笛卡儿积，它才会起作用。这样，自包含的 GROUP BY 查询就符合定义。两个例子都没有对 SQL 组的定义造成任何破坏只是为了完整起见才展示它们的，这样可以让读者意识到，在 SQL 中任何事都有可能发生。

## SELECT 和 GROUP BY 之间的关系

定义并证明了组概念的之后，现在可以转到更实际的应用上，即使用 GROUP BY 的查询。当在 SQL 中分组时，了解 SELECT 子句和 GROUP BY 子句之间的关系是非常重要的。一定要记住，当使用聚集函数（如 COUNT）时，对于 SELECT 列表中的项，如果没有把它当作聚集函数的参数使用，那么它必须是组的一部分。例如，有如下 SELECT 子句：

```

select deptno, count(*) as cnt
from emp

```

那么，一定要在 GROUP BY 子句中列出 DEPTNO：

```

select deptno, count(*) as cnt
from emp

```

group by deptno

DEPTNO	CNT
10	3
20	5
30	6

由用户定义的函数、窗口函数和非关联的标量子查询返回的常量、标量值，都是这种规则的例外。由于 SELECT 子句是在 GROUP BY 子句之后进行计算的，所以 SELECT 列表允许这些结构，而且不必（在某些情况下不能）在 GROUP BY 子句中指定。例如：

```
select 'hello' as msg,
       1 as num,
       deptno,
       (select count(*) from emp) as total,
       count(*) as cnt
from emp
group by deptno
```

MSG	NUM	DEPTNO	TOTAL	CNT
hello	1	10	14	3
hello	1	20	14	5
hello	1	30	14	6

不要被上述这个查询迷惑。SELECT 列表中没有包含在 GROUP BY 子句中的项不会影响各 DEPTNO 的 CNT 值，也不会改变 DEPTNO 值。基于上述查询的结果，当更明确地使用聚集时，现在可以对有关 SELECT 列表和 GROUP BY 子句中项目匹配的规则做个更准确的定义：

对于可能会更改组（或更改聚集函数返回的值）的 SELECT 列表项，一定包含于 GROUP BY 子句中。

上述 SELECT 列表中的附加项不会更改任何组（每个 DEPTNO）的 CNT 值，它们也不会更改组本身。

现在，可能有人要问：SELECT 列表中的哪些项会更改组及聚集函数返回的值呢？答案非常简单：SELECT 对象表的其他列。如果在查询中加入 JOB 列，如下所示：

```
select deptno, job, count(*) as cnt
from emp
group by deptno, job
```

DEPTNO	JOB	CNT
10	CLERK	1
10	MANAGER	1
10	PRESIDENT	1
20	CLERK	2
20	ANALYST	2
20	MANAGER	1
30	CLERK	1
30	MANAGER	1
30	SALESMAN	4

通过列出表 EMP 的另一列 JOB，表明要更改组，并更改结果集，因此，现在一定要在

GROUP BY 子句中包含 JOB 和 DEPTNO, 否则, 查询将会失败。SELECT/GROUP BY 子句中包含 JOB 后, 就把查询“每个部门有多少员工?”更改为“每个部门有多少不同职位的员工? ”。还要注意, 组是独特的, DEPTNO 和 JOB 的值都不是独特的, 但二者的组合 (它是 GROUP BY 和 SELECT 列表中的内容, 因此是组) 是独特的 (例如, 10 和 CLERK 的组合只出现一次)。

如果 SELECT 列表中只有聚集函数, 那么, 可以在 GROUP BY 子句中列出任意想要的有效列。下面两个查询验证了这个事实:

```
select count(*)
  from emp
 group by deptno

COUNT(*)
-----
3
5
6

select count(*)
  from emp
 group by deptno, job

COUNT(*)
-----
1
1
1
2
2
1
1
1
1
4
```

并不强制在 SELECT 列表中包含聚集函数以外的项, 但通常可提高结果的可读性和可用性。

**注意:** 通常, 在使用 GROUP BY 和聚集函数时, SELECT 列表[FROM 子句中的表]中的项, 如果没有用作聚集函数的参数, 那么一定要在 GROUP BY 子句中包含它们。然而, MySQL 有一个“特性”, 允许偏离这种规则, 允许把 SELECT 列表中没有用作聚集函数的参数的项[SELECT 对象表中的列], 也不必包含在 GROUP BY 子句中。这里“特性”这一术语使用很不严谨, 因为它就是随时会发生的缺陷, 所以应该避免使用它。事实上, 如果使用 MySQL, 并且非常关注查询的准确性的话, 应该删除这种“特征”。

## A.2 窗口

理解了在 SQL 中分组和使用聚集的概念后, 要理解窗口函数就非常容易了。与聚集函数一样, 窗口函数也针对定义的行集 (组) 执行聚集, 但它不像聚集函数那样每组只返回一个值, 窗口函数可以为每组返回多个值。执行聚集的行组是窗口 (因此命名为“窗口

函数”)。实际上, DB2 中称这种函数为联机分析处理 (OLAP) 函数, 而 Oracle 把它们称为解析函数, 但 ISO SQL 标准把它们称为窗口函数, 本书就使用这一术语。

## 一个简单的例子

要计算所有部门间的员工总数。传统方法是针对整个 EMP 表使用 COUNT(\*) 查询:

```
select count(*) as cnt
  from emp

      CNT
-----
      14
```

这是非常容易的, 但往往需要从不表示聚集或表示其他聚集的行中访问这种聚集数据。窗口函数解决了这种问题。例如, 下面的查询显示了如何使用窗口函数从细节行 (每个员工一行) 访问聚集数据 (员工总数):

```
select ename,
       deptno,
       count(*) over() as cnt
  from emp
 order by 2
```

ENAME	DEPTNO	CNT
CLARK	10	14
KING	10	14
MILLER	10	14
SMITH	20	14
ADAMS	20	14
FORD	20	14
SCOTT	20	14
JONES	20	14
ALLEN	30	14
BLAKE	30	14
MARTIN	30	14
JAMES	30	14
TURNER	30	14
WARD	30	14

在这个例子中, 窗口函数的调用格式为 COUNT(\*) OVER()。OVER 关键字表明: 把 COUNT 的调用看作窗口函数, 而不是聚集函数。总体上说, SQL 标准允许将所有聚集函数用作窗口函数, 而用关键字 OVER 区分两种用法。

窗口函数 COUNT(\*) OVER () 究竟都干了什么呢? 对于查询返回的每一行, 它返回了表中所有行的计数。空括号实际上是在暗示, OVER 关键字还可以接受其他子句, 以改变给定窗口函数作用的行范围。如果没有这样的子句, 窗口函数会看到结果集中所有的行, 这就是上面的输出中每行的值都是 14 的原因。

下面开始介绍窗口函数的大用途, 它们允许处理一行的多层聚集。学完本附录之后, 会明白它的用途多么难以置信。

## 计算顺序

在深入研究 OVER 子句之前，一定要注意：在 SQL 处理中，窗口函数都是最后一步执行，而且仅位于 ORDER BY 子句之前。作为验证窗口函数最后执行的例子，我们看一下上一节的查询，它使用 WHERE 子句从 DEPTNO 20 和 30 中筛选掉员工：

```
select  ename,
        deptno,
        count(*) over() as cnt
  from emp
 where deptno = 10
 order by 2
```

ENAME	DEPTNO	CNT
CLARK	10	3
KING	10	3
MILLER	10	3

每行的 CNT 值都不大于 14，现在它是 3。在这个例子中，WHERE 子句限制结果集显示三行，因此，窗口函数只计算三行（当处理到查询的 SELECT 部分时，窗口函数只能使用三行）。从这个例子中，可以看到，窗口函数是在子句（例如 WHERE 和 GROUP BY）之后进行计算的。

## 分区

使用 PARTITION BY 子句定义行的分区或组，以完成聚集。如果使用了空括号，那么整个结果集就是分区，窗口函数将针对它进行聚集计算。可以把 PARTITION BY 子句看作“移动的 GROUP BY”，这是因为与传统的 GROUP BY 不同，PARTITION BY 创建的组在结果集中并不是独特的。可以用 PARTITION BY 对定义的行组计算聚集（当遇到新组的时候复位），并返回每个值（每个组中的每个成员），而不是用一个组表示表中这个值的所有实例。请看下面的查询：

```
select  ename,
        deptno,
        count(*) over(partition by deptno) as cnt
  from emp
 order by 2
```

ENAME	DEPTNO	CNT
CLARK	10	3
KING	10	3
MILLER	10	3
SMITH	20	5
ADAMS	20	5
FORD	20	5
SCOTT	20	5
JONES	20	5
ALLEN	30	6
BLAKE	30	6
MARTIN	30	6
JAMES	30	6
TURNER	30	6
WARD	30	6



这个查询也会返回 14 行，但现在 PARTITION BY DEPTNO 子句的结果是为每个部门执行 COUNT，同一部门（同一个分区）的每个员工的 CNT 值相同，这是由于在遇到新部门之前不会重置（重新计算）聚集。也要注意，每个组的信息与及该组的每个成员同时返回。可以把上述查询看作是下列查询的更有效版本：

```
select e.ename,
       e.deptno,
       (select count(*) from emp d
        where e.deptno=d.deptno) as cnt
  from emp e
 order by 2
```

ENAME	DEPTNO	CNT
CLARK	10	3
KING	10	3
MILLER	10	3
SMITH	20	5
ADAMS	20	5
FORD	20	5
SCOTT	20	5
JONES	20	5
ALLEN	30	6
BLAKE	30	6
MARTIN	30	6
JAMES	30	6
TURNER	30	6
WARD	30	6

另外，PARTITION BY 子句的优点是：在同一个 SELECT 语句中，一个窗口函数的计算独立于按其他列分区的其他窗口函数的计算。参阅下列查询，它返回每个员工、他的部门、他的部门中的员工数、他的职位以及跟他相同职位的员工数：

```
select ename,
       deptno,
       count(*) over(partition by deptno) as dept_cnt,
       job,
       count(*) over(partition by job) as job_cnt
  from emp
 order by 2
```

ENAME	DEPTNO	DEPT_CNT	JOB	JOB_CNT
MILLER	10	3	CLERK	4
CLARK	10	3	MANAGER	3
KING	10	3	PRESIDENT	1
SCOTT	20	5	ANALYST	2
FORD	20	5	ANALYST	2
SMITH	20	5	CLERK	4
JONES	20	5	MANAGER	3
ADAMS	20	5	CLERK	4
JAMES	30	6	CLERK	4
MARTIN	30	6	SALESMAN	4
TURNER	30	6	SALESMAN	4
WARD	30	6	SALESMAN	4
ALLEN	30	6	SALESMAN	4
BLAKE	30	6	MANAGER	3

在这个结果集中，可以看到，同一部门中的员工的 DEPT\_CNT 值相同，而且职位相同的员工具有相同的 JOB\_CNT 值。

至此，应该很清楚，PARTITION BY 子句与 GROUP BY 子句的运行方式一样，只是它不受 SELECT 子句中的其他项影响，而且不需要编写 GROUP BY 子句。

## NULL 的作用

与 GROUP BY 子句一样，PARTITION BY 子句把所有的 NULL 合并为组或分区。这样，当使用 PARTITION BY 时，NULL 的作用与使用 GROUP BY 时 NULL 的作用相同。下面的查询使用窗口函数，计算各档提成的员工数（为使可读性更好，这里返回了 -1，而不是 NULL）：

```
select coalesce(comm,-1) as comm,
       count(*)over(partition by comm) as cnt
from emp
```

COMM	CNT
0	1
300	1
500	1
1400	1
-1	10
-1	10
-1	10
-1	10
-1	10
-1	10
-1	10
-1	10
-1	10
-1	10
-1	10

由于使用了 COUNT(\*), 因此对行进行计数。可以看到，有 10 名员工的佣金为 NULL。然而，如果不使用 \*，而是使用 COMM，就会得到另一个结果：

```
select coalesce(comm,-1) as comm,
       count(comm)over(partition by comm) as cnt
from emp
```

COMM	CNT
0	1
300	1
500	1
1400	1
-1	0
-1	0
-1	0
-1	0
-1	0
-1	0
-1	0
-1	0
-1	0
-1	0
-1	0

这个查询使用了 COUNT(COMM)，这就意味着只计算 COMM 列中的非 NULL 值。有个员工的佣金是 0，另一个员工的佣金是 300 等等。但要注意那些佣金是 NULL 员工！它们

的计数值是0。为什么呢？这是因为聚集函数忽略了NULL值，更确切地说，聚集函数只计算非NULL值。

注意：当使用COUNT时，需要考虑是否希望包含NULL。使用COUNT(column)，不会计算NULL。如果希望包含NULL，则使用COUNT(\*)（现在不再计数实际列值，而是对行计数）。

### 顺序何时会有影响

有时，窗口函数处理行的顺序很重要，会影响查询返回的结果。鉴于这种原因，窗口函数语法包含了可以放在OVER子句内的ORDER BY小子句。ORDER BY子句用于指定分区中行的排序方式（记住，如果PARTITION BY子句中没有“分区”，就意味着处理整个结果集）。

警告：有些窗口函数强制要求对分区中的行排序。因此，对于有些窗口函数，ORDER BY子句是必需的。

当在窗口函数的OVER子句中使用ORDER BY子句时，就指定了两件事：

- 1. 分区中的行如何排序
- 2. 在计算中包含哪些行

请看下面的查询，它计算了DEPTNO 10中员工的累加和：

```
select deptno,
       ename,
       hiredate,
       sal,
       sum(sal)over(partition by deptno) as total1,
       sum(sal)over() as total2,
       sum(sal)over(order by hiredate) as running_total
from emp
where deptno=10
```

DEPTNO	ENAME	HIREDATE	SAL	TOTAL1	TOTAL2	RUNNING_TOTAL
10	CLARK	09-JUN-1981	2450	8750	8750	2450
10	KING	17-NOV-1981	5000	8750	8750	7450
10	MILLER	23-JAN-1982	1300	8750	8750	8750

警告：为了便于理解，这里另包含了一个求和列，其OVER子句中用的是空括号。注意，TOTAL1和TOTAL2的值相同。为什么呢？窗口函数的计算顺序回答了这个问题。WHERE子句筛选出结果集，这样，总和只考虑DEPTNO 10中的工资。在这个例子中，只有一个分区，即整个结果集，它包含DEPTNO 10中的工资。因此TOTAL1和TOTAL2相同。

查看SAL列返回的值，很容易明白RUNNING\_TOTAL值的来源。可以观察这些值，自

已把它们加起来，计算累加和。但更重要的是，OVER 子句包含的 ORDER BY 为什么先创建累加和呢？其原因是：当在 OVER 子句中使用 ORDER BY 时，就会在分区内指定一个默认的“移动”或“滑动”窗口，即使看不到它。ORDER BY HIREDATE 子句也会在当前行的 HIREDATE 位置终止总和的计算。

下面的查询与前一个查询相同，只是使用了 RANGE BETWEEN 子句（后面会有更多的介绍）显式地指定 ORDER BY HIREDATE 产生的默认行为：

```
select deptno,
       ename,
       hiredate,
       sal,
       sum(sal)over(partition by deptno) as total1,
       sum(sal)over() as total2,
       sum(sal)over(order by hiredate
                    range between unbounded preceding
                    and current row) as running_total
from emp
where deptno=10
```

DEPTNO	ENAME	HIREDATE	SAL	TOTAL1	TOTAL2	RUNNING_TOTAL
10	CLARK	09-JUN-1981	2450	8750	8750	2450
10	KING	17-NOV-1981	5000	8750	8750	7450
10	MILLER	23-JAN-1982	1300	8750	8750	8750

这个查询中的 RANGE BETWEEN 子句被 ANSI 称为框架子句，这里也采用这种术语。现在，应该很容易理解在 OVER 子句中指定 ORDER BY 来创建累加和的原因；默认情况下会告诉查询：计算所有行的和，即从当前行开始、包括它前面的所有行（“前面的”是在 ORDER BY 中定义的，在这个例子中，行是按 HIREDATE 排序的）。

## 框架子句

现在在上述查询中使用框架子句，首先从员工 CLARK 开始。

1. 从 CLARK 的工资（2450）开始，而且包括在 CLARK 之前聘用的所有员工，再计算总和。由于 CLARK 是 DEPTNO 10 中聘用的第一个员工，所以，此时的总和就是 CLARK 的工资（2450），这是 RUNNING\_TOTAL 返回的第一个值。
2. 移动到下一个员工 KING（按 HIREDATE 排序），再一次使用框架子句。从当前行 5000（KING 的工资）开始计算 SAL 的和，其中包括他前面的所有行（KING 前面的所有员工）。CLARK 是唯一在 KING 之前聘用的员工，因此，和应该是 5000 + 2450，等于 7450，这就是 RUNNING\_TOTAL 返回的第二个值。
3. 移动到 MILLER，基于 HIREDATE 分区的最后一名员工，又一次使用框架子句。从当前行 1300（MILLER 的工资）开始计算 SAL 的和，其中包括他前面的所有行（MILLER 前面的所有员工）。CLARK 和 KING 是仅有在 MILLER 之前聘用的两名员工，因此，MILLER 的 RUNNING\_TOTAL 也包含他们的工资：2450+5000+1300，等于 8750，这就是 MILLER 的 RUNNING\_TOTAL 值。

可以看到，确实是框架子句生成了累加和。ORDER BY 定义了计算顺序，同时也暗示了默认框架。

通常，框架子句允许定义数据的不同“子窗口”，以便在计算中使用。有很多方式可以指定这样的子窗口。请看下列查询：

```
select deptno,
       ename,
       sal,
       sum(sal)over(order by hiredate
                    range between unbounded preceding
                          and current row) as run_total1,
       sum(sal)over(order by hiredate
                    rows between 1 preceding
                          and current row) as run_total2,
       sum(sal)over(order by hiredate
                    range between current row
                          and unbounded following) as run_total3,
       sum(sal)over(order by hiredate
                    rows between current row
                          and 1 following) as run_total4
from emp
where deptno=10
```

DEPTNO	ENAME	SAL	RUN_TOTAL1	RUN_TOTAL2	RUN_TOTAL3	RUN_TOTAL4
10	CLARK	2450	2450	2450	8750	7450
10	KING	5000	7450	7450	6300	6300
10	MILLER	1300	8750	6300	1300	1300

不要担心，这个查询并不像它看起来那么差。现在看到了RUN\_TOTAL1以及框架子句“UNBOUNDED PRECEDING AND CURRENT ROW”的效果。接下来，会说明在其他例子中发生的现象：

*RUN\_TOTAL2*

不同于关键字 RANGE，这个框架子句指定了 ROWS，这就意味着将通过计算行数来构造框架或窗口。1 PRECEDING 意味着框架将从当前行的前一行开始，其范围一直延续到 CURRENT ROW（当前行）。因此，得到的 RUN\_TOTAL2 就是当前员工的工资与前一个员工（按 HIREDATE）的工资和。

注意：非常凑巧，CLARK 和 KING 的 RUN\_TOTAL1 和 RUN\_TOTAL2 值都相同。为什么呢？回想一下，对于这两个窗口函数，它们为员工计算了哪些和值。仔细想想，就能得到答案。

*RUN\_TOTAL3*

RUN\_TOTAL3 窗口函数与 RUN\_TOTAL1 的作用相反，RUN\_TOTAL1 是从当前行开始，总和包括它前面的所有行，而 RUN\_TOTAL3 是从当前行开始，总和包括它后面的所有行。

### RUN\_TOTAL4

它与 RUN\_TOTAL2 相反, RUN\_TOTAL2 是从当前行开始, 总和中包括它前面的一行, 而 RUN\_TOTAL4 是从当前行开始, 总和中包括它后面的一行。

**注意:** 如果理解了上述解释, 那么阅读本书的任何内容都不成问题。如果还是不理解, 试着用自己的例子和数据进行实验。自己编写代码体验新特性, 而不仅停留在阅读上, 那么学习起来会更容易。

## 关于框架的最后讨论

最后一个例子, 展示了框架子句对查询输出的影响, 请看下列查询:

```
select ename,
       sal,
       min(sal)over(order by sal) min1,
       max(sal)over(order by sal) max1,
       min(sal)over(order by sal
                    range between unbounded preceding
                    and unbounded following) min2,
       max(sal)over(order by sal
                    range between unbounded preceding
                    and unbounded following) max2,
       min(sal)over(order by sal
                    range between current row
                    and current row) min3,
       max(sal)over(order by sal
                    range between current row
                    and current row) max3,
       max(sal)over(order by sal
                    rows between 3 preceding
                    and 3 following) max4
from emp
```

ENAME	SAL	MIN1	MAX1	MIN2	MAX2	MIN3	MAX3	MAX4
SMITH	800	800	800	800	5000	800	800	1250
JAMES	950	800	950	800	5000	950	950	1250
ADAMS	1100	800	1100	800	5000	1100	1100	1300
WARD	1250	800	1250	800	5000	1250	1250	1500
MARTIN	1250	800	1250	800	5000	1250	1250	1600
MILLER	1300	800	1300	800	5000	1300	1300	2450
TURNER	1500	800	1500	800	5000	1500	1500	2850
ALLEN	1600	800	1600	800	5000	1600	1600	2975
CLARK	2450	800	2450	800	5000	2450	2450	3000
BLAKE	2850	800	2850	800	5000	2850	2850	3000
JONES	2975	800	2975	800	5000	2975	2975	5000
SCOTT	3000	800	3000	800	5000	3000	3000	5000
FORD	3000	800	3000	800	5000	3000	3000	5000
KING	5000	800	5000	800	5000	5000	5000	5000

好吧, 下面分析这个查询:

### MIN1

生成这个列的窗口函数没有指定框架子句, 因此, UNBOUNDED PRECEDING AND CURRENT ROW 的默认框架子句“破门而入”。为什么所有行的 MIN1 都是



800呢？这是因为最低工资是先出现的（ORDER BY SAL），而且此后它一直是最低的（或最小的）工资。

#### MAX1

MAX1 值不同于 MIN1。为什么？答案还是默认框架子句 UNBOUNDED PRECEDING AND CURRENT ROW。该框架子句与 ORDER BY SAL 一起使用，确保最高工资也总是当前行的工资。

观察第一行，即 SMITH。当计算 SMITH 的工资及所有以前的工资时，SMITH 的 MAX1 就是 SMITH 的工资，原因是他前面没有其他的工资。再看看下一行，即 JAMES，当把 JAMES 的工资与前面的所有工资相比较时，在这个例子中，与 SMITH 的工资相比较，则 JAMES 的工资更高，因此，它就是最大值。将这种逻辑应用于所有行，就会看到每行的 MAX1 值都是当前员工的工资。

#### MIN2 和 MAX2

给定的框架子句是 UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING，这与指定空括号一样。因此，当计算 MIN 和 MAX 时，会考虑结果集中的所有行。可能希望整个结果集的 MIN 和 MAX 值都是常量，所以这些列的值也是常量。

#### MIN3 和 MAX3

二者的框架子句是 CURRENT ROW AND CURRENT ROW，这就意味着：当寻找 MIN 和 MAX 工资时只需使用当前员工的工资。MIN3 和 MAX3 值与每行的 SAL 相同。这很容易理解，不是吗？

#### MAX4

为 MAX4 定义的框架子句是 3 PRECEDING AND 3 FOLLOWING，这就意味着：对每一行，都要考虑当前行、它的前三行和它的后三行。对该特定 MAX(SAL) 的调用会从这些行返回最高工资值。

如果查看员工 MARTIN 的 MAX4 值，就会看到如何应用框架子句。MARTIN 的工资是 1250，MARTIN 前面的三个员工的工资分别为 WARD 的工资（1250）、ADAMS 的工资（1100）以及 JAMES 的工资（950）。MARTIN 后面的三个员工的工资分别为 MILLER 的工资（1300）、TURNER 的工资（1500）以及 ALLEN 的工资（1600）。除了这些工资之外，还包括 MARTIN 自己的工资，最高工资是 ALLEN 的工资，因此，MAX4 值是 MARTIN 的 1600。

## A.3 可读性 + 性能 = 威力

可以看到，窗口函数的功能非常强大，它们允许编写包含细目和聚集信息的查询。使用窗口函数，与使用多个自联接和/或标量子查询相比较，它需要编写的查询更小、更有效。

请看下面的查询，它很容易地回答了如下问题：“每个部门的员工数是多少？每个部门中有多少种不同职位的员工（例如，部门 10 中有多少个办事员）？表 EMP 中的员工总数是多少？”

```
select deptno,
       job,
       count(*) over (partition by deptno) as emp_cnt,
       count(job) over (partition by deptno,job) as job_cnt,
       count(*) over () as total
from emp
```

DEPTNO	JOB	EMP_CNT	JOB_CNT	TOTAL
10	CLERK	3	1	14
10	MANAGER	3	1	14
10	PRESIDENT	3	1	14
20	ANALYST	5	2	14
20	ANALYST	5	2	14
20	CLERK	5	2	14
20	CLERK	5	2	14
20	MANAGER	5	1	14
30	CLERK	6	1	14
30	MANAGER	6	1	14
30	SALESMAN	6	4	14
30	SALESMAN	6	4	14
30	SALESMAN	6	4	14
30	SALESMAN	6	4	14

如果不使用窗口函数，要返回同样的结果集，需要额外多做一点儿工作：

```
select a.deptno, a.job,
       (select count(*) from emp b
        where b.deptno = a.deptno) as emp_cnt,
       (select count(*) from emp b
        where b.deptno = a.deptno and b.job = a.job) as job_cnt,
       (select count(*) from emp) as total
from emp a
order by 1,2
```

DEPTNO	JOB	EMP_CNT	JOB_CNT	TOTAL
10	CLERK	3	1	14
10	MANAGER	3	1	14
10	PRESIDENT	3	1	14
20	ANALYST	5	2	14
20	ANALYST	5	2	14
20	CLERK	5	2	14
20	CLERK	5	2	14
20	MANAGER	5	1	14
30	CLERK	6	1	14
30	MANAGER	6	1	14
30	SALESMAN	6	4	14
30	SALESMAN	6	4	14
30	SALESMAN	6	4	14
30	SALESMAN	6	4	14

很显然，非窗口解决方案并不难编写，但它不是很整洁、高效（对于一个 14 行表，不会看到性能差异，但对一个 1000 行或 10000 行表试验这些查询，那么会看到使用窗口函数比采用多个自联接和标量子查询有更多益处）。



## A.4 提供一个基石

除了可读性和性能之外，窗口函数可用于为更复杂的“报表样式”查询提供一个“基石”。例如，考虑下面的“报表样式”查询，它在内联视图中使用了窗口函数，然后在外层查询中计算结果的总和。使用窗口函数，可以返回细目及聚集数据，这对报表是非常有用的。下面的查询使用窗口函数查找不同分区的计数。由于聚集应用于多行，所以内联视图会返回 EMP 中的所有行，而外层 CASE 表达式可以使用它转换和创建格式化的报表：

```
select deptno,
       emp_cnt as dept_total,
       total,
       max(case when job = 'CLERK'
                then job_cnt else 0 end) as clerks,
       max(case when job = 'MANAGER'
                then job_cnt else 0 end) as mgrs,
       max(case when job = 'PRESIDENT'
                then job_cnt else 0 end) as prez,
       max(case when job = 'ANALYST'
                then job_cnt else 0 end) as anals,
       max(case when job = 'SALESMAN'
                then job_cnt else 0 end) as smen
from (
select deptno,
       job,
       count(*) over (partition by deptno) as emp_cnt,
       count(job) over (partition by deptno, job) as job_cnt,
       count(*) over () as total
from emp
) x
group by deptno, emp_cnt, total
```

DEPTNO	DEPT_TOTAL	TOTAL	CLERKS	MGRS	PREZ	ANALS	SMEN
10	3	14	1	1	1	0	0
20	5	14	2	1	0	2	0
30	6	14	1	1	0	0	4

上面的查询返回了每个部门、各部门的员工数、表 EMP 中的员工总数以及每个部门中不同职位的细目。所有这些都是在一个查询内完成的，不需要额外联接，也不需要使用临时表！

最后一个例子证明了：如果使用窗口函数，很多问题都很容易回答，请看下面的查询：

```
select ename as name,
       sal,
       max(sal) over (partition by deptno) as hiDpt,
       min(sal) over (partition by deptno) as loDpt,
       max(sal) over (partition by job) as hiJob,
       min(sal) over (partition by job) as loJob,
       max(sal) over () as hi,
       min(sal) over () as lo,
       sum(sal) over (partition by deptno
                    order by sal, empno) as dptRT,
       sum(sal) over (partition by deptno) as dptSum,
       sum(sal) over () as ttl
from emp
order by deptno, dptRT
```

NAME	SAL	HIDPT	LODPT	HIJOB	LOJOB	HI	LO	DPTRT	DPTSUM	TTL
MILLER	1300	5000	1300	1300	800	5000	800	1300	8750	29025
CLARK	2450	5000	1300	2975	2450	5000	800	3750	8750	29025
KING	5000	5000	1300	5000	5000	5000	800	8750	8750	29025
SMITH	800	3000	800	1300	800	5000	800	800	10875	29025
ADAMS	1100	3000	800	1300	800	5000	800	1900	10875	29025
JONES	2975	3000	800	2975	2450	5000	800	4875	10875	29025
SCOTT	3000	3000	800	3000	3000	5000	800	7875	10875	29025
FORD	3000	3000	800	3000	3000	5000	800	10875	10875	29025
JAMES	950	2850	950	1300	800	5000	800	950	9400	29025
WARD	1250	2850	950	1600	1250	5000	800	2200	9400	29025
MARTIN	1250	2850	950	1600	1250	5000	800	3450	9400	29025
TURNER	1500	2850	950	1600	1250	5000	800	4950	9400	29025
ALLEN	1600	2850	950	1600	1250	5000	800	6550	9400	29025
BLAKE	2850	2850	950	2975	2450	5000	800	9400	9400	29025

这个查询回答了下列问题，它非常简单、高效，而且可读性更好（没有对EMP进行额外联接！）。只要把员工及其工资与结果集中的不同行相匹配，就可以确定：

1. 所有员工中谁的工资最高（HI）
2. 所有员工中谁的工资最低（LO）
3. 本部门中谁的工资最高（HIDPT）
4. 本部门中谁的工资最低（LODPT）
5. 同一职务中谁的工资最高（HIJOB）
6. 同一职务中谁的工资最低（LOJOB）
7. 所有工资的总和是多少（TTL）
8. 每个部门工资的总和是多少（DPTSUM）
9. 每个部门内工资的累加和是多少（DPTRT）

# 回顾 Rozenshtein

谨以此附录献给 David Rozenshtein。在引言中曾经提到过，他的书《The Essence of SQL》是（迄今为止）有关 SQL 内容的最佳图书。尽管只有 119 页，但书中介绍的内容，都是 SQL 程序员应该了解的至关重要的话题。特别的是 David 说明了如何思考问题以及如何得出答案。Rozenshtein 提供的解决方案都是面向集合的。在实际环境中，即使表的大小不允许采纳他的解决方案，但他的方法还是可取的，可以让程序员停止寻求过程化的解决方案，并开始以集合的概念思考。

《The Essence of SQL》是在窗口函数和 MODEL 子句出现之前出版的。在本附录中，针对 Rozenshtein 的书中的某些问题，使用 SQL 标准提供的较新的函数，给出了另外一种解决方案（新解决方案是否比 Rozenshtein 的方案“更好”，取决于具体环境）。在每个讨论的末尾，都基于这本书原始解决方案提出了其他的解决方案。例如，本章可能根据 Rozenshtein 书中的问题提出一个变种，也会给出相应解决方案的变种（Rozenshtein 的书可能没有必要列出这种解决方案，但它们采用了相似的技巧）。

## B.1 Rozenshtein 的用例表

下面的表是以 Rozenshtein 的书为基础，本章将使用它：

```
/* table of students */
create table student
( sno      integer,
  sname    varchar(10),
  age      integer
)

/* table of courses */
create table courses
( cno      varchar(5),
  title    varchar(10),
  credits  integer
)

/* table of professors */
create table professor
( lname    varchar(10),
```

```

dept    varchar(10),
salary integer,
age     integer
)

/* table of students and the courses they take */
create table take
( sno     integer,
  cno     varchar(5)
)

/* table of professors and the courses they teach */
create table teach
( lname   varchar(10),
  cno     varchar(5)
)

insert into student values (1,'AARON',20)
insert into student values (2,'CHUCK',21)
insert into student values (3,'DOUG',20)
insert into student values (4,'MAGGIE',19)
insert into student values (5,'STEVE',22)
insert into student values (6,'JING',18)
insert into student values (7,'BRIAN',21)
insert into student values (8,'KAY',20)
insert into student values (9,'GILLIAN',20)
insert into student values (10,'CHAD',21)

insert into courses values ('CS112','PHYSICS',4)
insert into courses values ('CS113','CALCULUS',4)
insert into courses values ('CS114','HISTORY',4)

insert into professor values ('CHOI','SCIENCE',400,45)
insert into professor values ('GUNN','HISTORY',300,60)
insert into professor values ('MAYER','MATH',400,55)
insert into professor values ('POMEL','SCIENCE',500,65)
insert into professor values ('FEUER','MATH',400,40)

insert into take values (1,'CS112')
insert into take values (1,'CS113')
insert into take values (1,'CS114')
insert into take values (2,'CS112')
insert into take values (3,'CS112')
insert into take values (3,'CS114')
insert into take values (4,'CS112')
insert into take values (4,'CS113')
insert into take values (5,'CS113')
insert into take values (6,'CS113')
insert into take values (6,'CS114')

insert into teach values ('CHOI','CS112')
insert into teach values ('CHOI','CS113')
insert into teach values ('CHOI','CS114')
insert into teach values ('POMEL','CS113')
insert into teach values ('MAYER','CS112')
insert into teach values ('MAYER','CS114')

```

## B.2 回答关于否定的问题

Rozenshtein 在书中，通过测验经常需要解决的各种类型的基本问题的方法来讲述 SQL。否定就是其中一种类型。通常需要找到不满足某种条件的行，对简单条件是相当容易的，但正如下列问题所显示的那样，有些否定问题需要一点儿创新思维才能解决。

## 问题 1

找到没有选择 CS112 课程的学生，下面的查询会返回错误结果：

```
select *
  from student
 where sno in ( select sno
                from take
                where cno != 'CS112' )
```

由于学生可以选取几门课程，上述查询就可能（真的会）返回选取了 CS112 课程的学生。这个查询错误的原因，是它并没有回答问题：“谁没有选取 CS112？”然而，它回答了如下问题：“谁选取了不是 CS112 的课程？”。正确的结果集应该包含没有选取任何课程的学生以及选取除 CS112 之外的其他课程的学生。最后，应该返回下列结果集：

SNO	SNAME	AGE
5	STEVE	22
6	JING	18
7	BRIAN	21
8	KAY	20
9	GILLIAN	20
10	CHAD	21

## MySQL 和 PostgreSQL

使用 CASE 表达式及聚集函数 MAX，对选取 CS112 课程的特殊学生做标记：

```
1 select s.sno,s.sname,s.age
2   from student s left join take t
3     on (s.sno = t.sno)
4   group by s.sno,s.sname,s.age
5   having max(case when t.cno = 'CS112'
6                 then 1 else 0 end) = 0
```

## DB2 和 SQL Server

使用 CASE 表达式及窗口函数 MAX OVER，对选取 CS112 课程的特殊学生做标记：

```
1 select distinct sno,sname,age
2   from (
3 select s.sno,s.sname,s.age,
4        max(case when t.cno = 'CS112'
5                then 1 else 0 end)
6        over(partition by s.sno,s.sname,s.age) as takes_CS112
7   from student s left join take t
8     on (s.sno = t.sno)
9   ) x
10  where takes_CS112 = 0
```

## Oracle

对于 Oracle9i Database 及更高版本的用户，可以采用上述的 DB2 解决方案。另外，还可以利用 Oracle 外联接语法，对于 Oracle8i Database 或较早版本的用户，只能采用这种方案：

```
/* group by solution */
```

```

1  select s.sno,s.sname,s.age
2     from student s, take t
3     where s.sno = t.sno (+)
4     group by s.sno,s.sname,s.age
5     having max(case when t.cno = 'CS112'
6                 then 1 else 0 end) = 0

/* window solution */

1  select distinct sno,sname,age
2     from (
3     select s.sno,s.sname,s.age,
4            max(case when t.cno = 'CS112'
5                    then 1 else 0 end)
6            over(partition by s.sno,s.sname,s.age) as takes_CS112
7     from student s, take t
8     where s.sno = t.sno (+)
9     ) x
10    where takes_CS112 = 0

```

## 讨论

尽管每种解决方案采用了不同的语法，但其技巧是相同的。在结果集中创建一个“布尔”列，用于表示学生是否选取了CS112课程。如果某个学生选取了CS112课程，那么该列就返回1，否则返回0。下面的查询把CASE表达式移到了SELECT列表中，显示出了中间结果：

```

select s.sno,s.sname,s.age,
       case when t.cno = 'CS112'
            then 1
            else 0
       end as takes_CS112
from student s left join take t
on (s.sno=t.sno)

```

SNO	SNAME	AGE	TAKES_CS112
1	AARON	20	1
1	AARON	20	0
1	AARON	20	0
2	CHUCK	21	1
3	DOUG	20	1
3	DOUG	20	0
4	MAGGIE	19	1
4	MAGGIE	19	0
5	STEVE	22	0
6	JING	18	0
6	JING	18	0
8	KAY	20	0
10	CHAD	21	0
7	BRIAN	21	0
9	GILLIAN	20	0

对表TAKE的外联接确保返回那些未选取任何课程的学生。下一步，使用MAX，获取CASE表达式为每个学生返回的最大值。如果某个学生选取了CS112课程，那么最大值就是1，原因是其他课程都为0。对于采用GROUP BY的解决方案，最后一步是使用HAVING子句，仅保留MAX/CASE表达式返回0值的学生。对于窗口解决方案，需要把该查询包入内联视图中，然后引用TAKES\_CS112，因为在WHERE子句中不能直接引用窗口函数。由于窗口函数工作机理的缘故，需要排除选多个课程引起的重复。

## 原始解决方案

这个问题的原始解决方案非常灵巧，如下所示：

```
select *
  from student
 where sno not in (select sno
                   from take
                   where cno = 'CS112')
```

可以把上述语句解释为：“从表 TAKE 中找到选取了 CS112 课程的学生，然后，返回表 STUDENT 中除这些学生之外的所有学生”。这种技巧采纳了 Rozenshtein 的书的结尾部分有关否定的建议：

记住，真正的否定需要两步操作：要找出“谁不在”，需先找出“谁在”，然后排除它们。

## 问题 2

找到选取 CS112 或选取 CS114（但不是二者）的学生。下面的查询看起来解决了问题，但返回了错误的结果集：

```
select *
  from student
 where sno in ( select sno
                from take
                where cno != 'CS112'
                and cno != 'CS114' )
```

所有选取了课程的学生中，只有 DOUG 和 AARON 同时选取了 CS112 和 CS114。这两个学生应该排除在外。学生 STEVE 选取了 CS113 课程，但没有选取 CS112 或 CS114，也应该排除在外。

由于学生可以选取多门课程，因此这里的方法是为每个学生返回一行信息，指明学生选取了 CS112、CS114 还是二者都选了。采用这种方法，就可以很容易地计算学生是否同时选取了这两门课程，而不必多次处理数据。最终结果集应该如下：

SNO	SNAME	AGE
2	CHUCK	21
4	MAGGIE	19
6	JING	18

## MySQL 和 PostgreSQL

使用 CASE 表达式及聚集函数 SUM，找到选取了 CS112 或 CS114（但不是二者）的学生：

```
1 select s.sno,s.sname,s.age
2   from student s, take t
3  where s.sno = t.sno
4   group by s.sno,s.sname,s.age
5  having sum(case when t.cno in ('CS112','CS114')
6                then 1 else 0 end) = 1
```

## DB2、Oracle 和 SQL Server

使用 CASE 表达式及窗口函数 SUM OVER，找到选取了 CS112 或 CS114（但不是二者）的学生：

```

1 select distinct sno,sname,age
2   from (
3 select s.sno,s.sname,s.age,
4        sum(case when t.cno in ('CS112','CS114') then 1 else 0 end)
5        over (partition by s.sno,s.sname,s.age) as takes_either_or
6   from student s, take t
7  where s.sno = t.sno
8        ) x
9  where takes_either_or = 1

```

## 讨论

解决这个问题的第一步是采用从表 STUDENT 到表 TAKE 的内联接，这样，就去掉了没有选取任何课程的学生；下一步，使用 CASE 表达式，指明学生是否选取了其中的一门课程。在下面的查询中，把 CASE 表达式移到了 SELECT 列表内，以返回中间结果：

```

select s.sno,s.sname,s.age,
       case when t.cno in ('CS112','CS114')
            then 1 else 0 end as takes_either_or
  from student s, take t
 where s.sno = t.sno

```

SNO	SNAME	AGE	TAKES_EITHER_OR
1	AARON	20	1
1	AARON	20	0
1	AARON	20	1
2	CHUCK	21	1
3	DOUG	20	1
3	DOUG	20	1
4	MAGGIE	19	1
4	MAGGIE	19	0
5	STEVE	22	0
6	JING	18	0
6	JING	18	1

TAKES\_EITHER\_OR 的值为 1 说明学生选取了 CS112 或 CS114。由于学生可以选取多门课程，因此，下一步使用 GROUP BY 及聚集函数 SUM，为每个学生返回一行。函数 SUM 将计算每个学生所有 1 的总和：

```

select s.sno,s.sname,s.age,
       sum(case when t.cno in ('CS112','CS114')
            then 1 else 0 end) as takes_either_or
  from student s, take t
 where s.sno = t.sno
 group by s.sno,s.sname,s.age

```

SNO	SNAME	AGE	TAKES_EITHER_OR
1	AARON	20	2
2	CHUCK	21	1
3	DOUG	20	2
4	MAGGIE	19	1
5	STEVE	22	0
6	JING	18	1



对于没有选取 CS112 或 CS114 的学生, 他的 TAKES\_EITHER\_OR 值为 0。选取了 CS112 和 CS114 的学生, 他的 TAKES\_EITHER\_OR 值为 2。这样, 只需返回 TAKES\_EITHER\_OR 值为 1 的学生。最终解决方案使用 HAVING 子句, 保留 TAKES\_EITHER\_OR 的 SUM (和) 为 1 的学生。

对于窗口解决方案, 采用了相同的技巧。也需要把查询包入内联视图中, 然后引用列 TAKES\_EITHER\_OR, 因为 WHERE 子句中不能直接引用窗口函数 (在 SQL 处理中, 它们仅在 ORDER BY 子句之前进行计算的)。由于窗口函数工作机理的缘故, 需要排除选多个课程引起的重复。

### 原始解决方案

下面的查询是原始解决方案 (稍微修改)。这个查询相当灵巧, 也采用了与问题 1 中原始解决方案相同的方法。该解决方案使用自联接查找选取 CS112 和 CS114 课程的学生, 然后使用子查询, 筛选出选取 CS112 或 CS114 的学生:

```
select *
  from student s, take t
 where s.sno = t.sno
    and t.cno in ( 'CS112', 'CS114' )
    and s.sno not in ( select a.sno
                      from take a, take b
                     where a.sno = b.sno
                       and a.cno = 'CS112'
                       and b.cno = 'CS114' )
```

### 问题 3

查找选取了 CS112 而且未选取其他课程的学生, 但下面的查询返回了错误结果:

```
select s.*
  from student s, take t
 where s.sno = t.sno
    and t.cno = 'CS112'
```

CHUCK 是唯一选取 CS112 而未选取其他课程的学生, 他是查询应该返回的唯一学生。

这个问题换一种说法 “找到只选取 CS112 课程的学生。” 上面的查询找到了选取 CS112 课程的学生, 但也返回了选取其他课程的学生。查询应该回答如下问题 “谁只选取了一门课程, 而且这个课程是 CS112?”

### MySQL 和 PostgreSQL

使用聚集函数 COUNT, 确保查询返回的学生只有一门课程:

```
1 select s.*
2   from student s,
3        take t1,
4        (
5 select sno
6   from take
7  group by sno
```

```

8  having count(*) = 1
9  ) t2
10 where s.sno = t1.sno
11      and t1.sno = t2.sno
12      and t1.cno = 'CS112'

```

## DB2、Oracle 和 SQL Server

使用窗口函数 COUNT OVER，确保学生只选取一门课程：

```

1  select sno,sname,age
2  from (
3  select s.sno,s.sname,s.age,t.cno,
4         count(t.cno) over (
5             partition by s.sno,s.sname,s.age
6             ) as cnt
7  from student s, take t
8  where s.sno = t.sno
9  ) x
10 where cnt = 1
11      and cno = 'CS112'

```

## 讨论

该解决方案的关键是编写一个查询，回答下面两个问题：“哪些学生只选取了一门课程？”以及“哪个学生选取了 CS112 课程？”第一种方法是使用内联视图 T2，找到只选取一门课程的学生。下一步，把内联视图 T2 联接到表 TAKE，并保留选取 CS112 课程的学生（现在，剩下的就是只选取了一门课程、而且这个课程是 CS112 的学生）。下面的查询显示了结果：

```

select t1.*
  from take t1,
  (
select sno
  from take
 group by sno
 having count(*) = 1
  ) t2
 where t1.sno = t2.sno
    and t1.cno = 'CS112'

SNO CNO
----
2 CS112

```

最后一步，联接表 STUDENT，在内联视图 T2 和表 TAKE 的返回行中，查找相匹配的学生。窗口解决方案采用了相似的方案，只是方式不同（更有效）。内联视图 X 返回了学生、他们选取的课程以及选取的课程数（表 TAKE 和表 STUDENT 之间的内联接确保把未选课程的学生排除）。其结果如下所示：

```

select s.sno,s.sname,s.age,t.cno,
       count(t.cno) over (
           partition by s.sno,s.sname,s.age
           ) as cnt
  from student s, take t
 where s.sno = t.sno

```

SNO	SNAME	AGE	CNO	CNT
1	AARON	20	CS112	3
1	AARON	20	CS113	3
1	AARON	20	CS114	3
2	CHUCK	21	CS112	1
3	DOUG	20	CS112	2
3	DOUG	20	CS114	2
4	MAGGIE	19	CS112	2
4	MAGGIE	19	CS113	2
5	STEVE	22	CS113	1
6	JING	18	CS113	2
6	JING	18	CS114	2

有了课程及计数之后，最后一步只要保留 CNT 是 1 并且 CNO 是 CS112 的学生即可。

### 原始解决方案

原始解决方案使用了子查询及两个否定：

```
select s.*
  from student s, take t
 where s.sno = t.sno
    and s.sno not in ( select sno
                      from take
                      where cno != 'CS112' )
```

这是非常灵巧的解决方案，由于查询中没有检查课程数，而且也没有筛选出选取了 CS112 的学生！那么，这如何起作用呢？该子查询返回选取一门不是 CS112 课程的所有学生，其结果如下所示：

```
select sno
  from take
 where cno != 'CS112'
```

```
SNO
----
1
1
3
4
5
6
6
```

外层查询返回选取一门课程（任意课程）的所有学生，这些学生并不属于子查询返回的学生。如果暂时忽略外部查询中的 NOT IN，结果将如下所示（选取一门课程的所有学生）：

```
select s.*
  from student s, take t
 where s.sno = t.sno
```

```
SNO SNAME      AGE
-----
1  AARON        20
1  AARON        20
1  AARON        20
2  CHUCK        21
3  DOUG         20
3  DOUG         20
```

4	MAGGIE	19
4	MAGGIE	19
5	STEVE	22
6	JING	18
6	JING	18

如果比较两个结果集，会看到外层查询的附加部分 NOT IN 有效地执行了外层查询的 SNO 和子查询的 SNO 之间的集合差值，只返回 SNO 值为 2 的学生。总之，子查询会找到所有选取了一门不是 CS112 课程的学生。外层查询返回不属于上述子查询输出的所有学生（这时，剩下的学生是选取了 CS112 或未选取任何课程的学生）。表 STUDENT 和表 TAKE 之间的联接会筛选掉未选取任何课程的学生，现在剩下的就是仅选取了一门 CS112 课程的学生。这样，就以最佳方式解决了基于集合的问题！

## B.3 回答有关“至多”的问题

涉及“至多”的问题是另一种类型的查询问题，有时可能会遇到这种问题。要找到满足条件的行相当容易，但假设给这些行添加一个限制条件，会怎么样呢？下面的两个问题将介绍这方面的内容。

### 问题 4

找到至多选取两门课程的学生，没有选取任何课程的学生应该排除掉。在选取了课程的学生中，只有 AARON 选取的课程多于两个，应该从结果集中删掉他。最后，返回下列结果集：

SNO	SNAME	AGE
2	CHUCK	21
3	DOUG	20
4	MAGGIE	19
5	STEVE	22
6	JING	18

## MySQL 和 PostgreSQL

使用聚集函数 COUNT，确定哪些学生选取的课程不多于两门：

```

1 select s.sno,s.sname,s.age
2   from student s, take t
3  where s.sno = t.sno
4  group by s.sno,s.sname,s.age
5 having count(*) <= 2

```

## DB2、Oracle 和 SQL Server

使用窗口函数 COUNT OVER，确定哪些学生选取的课程不多于两门：

```

1 select distinct sno,sname,age
2   from (
3 select s.sno,s.sname,s.age,
4        count(*) over (
5          partition by s.sno,s.sname,s.age
6          ) as cnt

```

```

7   from student s, take t
8   where s.sno = t.sno
9       ) x
10  where cnt <= 2

```

## 讨论

两种解决方案都是只要计算特定的SNO在表TAKE中出现的次数。与表TAKE的内联接确保从最终结果集中删除未选取任何课程的学生。

## 原始解决方案

Rozenshtein采用了上述MySQL和PostgreSQL中的聚集解决方案，他也介绍了另一种使用多次自联接的解决方案，如下所示：

```

select distinct s.*
  from student s, take t
 where s.sno = t.sno
    and s.sno not in ( select t1.sno
                      from take t1, take t2, take t3
                     where t1.sno = t2.sno
                       and t2.sno = t3.sno
                       and t1.cno < t2.cno
                       and t2.cno < t3.cno )

```

多次自联接的解决方案很有趣，它无需使用聚集就解决了问题。要了解该解决方案的运行方式，重点研究子查询的WHERE子句即可。按SNO的内联接确保了在子查询返回的每行的来自各个表的所有列都是同一个学生的信息。小于比较操作确定了学生是否选取了两门以上的课程。子查询中的WHERE子句可以解释为：“对于某个特殊学生，返回满足下述条件的行：即他的第一个CNO小于第二个CNO、而且第二个CNO小于第三个CNO。”如果学生选取的课程少于三门，则该表达式永远不会为真，原因是并不存在第三个CNO。子查询的目的是找到选取三门以上课程的学生。然后，外层查询返回具有如下条件的学生：至少选取一门课程，而且他不属于子查询返回的结果。

## 问题 5

找到至多比另外两名学生大的学生。换一种说法是找到比0个学生、一个学生、两个学生大的那些学生。最终结果集应该如下：

SNO	SNAME	AGE
6	JING	18
4	MAGGIE	19
1	AARON	20
9	GILLIAN	20
8	KAY	20
3	DOUG	20

## MySQL 和 PostgreSQL

使用聚集函数COUNT和关联的子查询，找到比0个学生、一个学生、两个学生大的那些学生：

```

1 select s1.*
2   from student s1
3  where 2 >= ( select count(*)
4               from student s2
5               where s2.age < s1.age )

```

## DB2、Oracle 和 SQL Server

使用窗口函数 DENSE\_RANK，找到比 0 个学生、一个学生、两个学生大的那些学生：

```

1 select sno,sname,age
2   from (
3 select sno,sname,age,
4        dense_rank()over(order by age) as dr
5   from student
6   ) x
7  where dr <= 3

```

## 讨论

聚集解决方案使用标量子查询，查找不比另外两个学生大的所有学生。要了解如何实现这种功能，需重新编写一个使用标量子查询的解决方案。在下面的例子中，CNT 列表示比当前学生年轻的学生数：

```

select s1.*,
       (select count(*) from student s2
        where s2.age < s1.age) as cnt
  from student s1
 order by 4

```

SNO	SNAME	AGE	CNT
6	JING	18	0
4	MAGGIE	19	1
1	AARON	20	2
3	DOUG	20	2
8	KAY	20	2
9	GILLIAN	20	2
2	CHUCK	21	6
7	BRIAN	21	6
10	CHAD	21	6
5	STEVE	22	9

按这种方式重新编写解决方案，在最终结果集中查看 CNT 小于等于 2 的学生就非常容易。

使用窗口函数 DENSE\_RANK 的解决方案与标量子查询例子非常相似，按比当前学生年轻的学生数给每一行分等级（允许捆绑，而且没有缝隙）。下面的查询显示了 DENSE\_RANK 函数的输出：

```

select sno,sname,age,
       dense_rank()over(order by age) as dr
  from student

```

SNO	SNAME	AGE	DR
6	JING	18	1
4	MAGGIE	19	2
1	AARON	20	3
3	DOUG	20	3
8	KAY	20	3
9	GILLIAN	20	3

2	CHUCK	21	4
7	BRIAN	21	4
10	CHAD	21	4
5	STEVE	22	5

最后一步，把该查询包入内联视图，只保留 DR 小于等于 3 的那些行。

### 原始解决方案

Rozenshtein 采用了一个有趣的方法解决这个问题。不是“找到至多比两个学生大的学生”，他的方法是“找到不比三个以上（至少三个）学生大的学生”。对于想学习如何用集合解决问题的程序员来说，这种方法非常完美，原因是它强制分两步实现解决方案：

1. 找到比三个以上学生大的学生集。
2. 只要返回不属于步骤 1 结果集的所有学生。

该解决方案如下所示：

```
select *
  from student
 where sno not in (
select s1.sno
  from student s1,
       student s2,
       student s3,
       student s4
 where s1.age > s2.age
       and s2.age > s3.age
       and s3.age > s4.age
)
```

SNO	SNAME	AGE
6	JING	18
4	MAGGIE	19
1	AARON	20
9	GILLIAN	20
8	KAY	20
3	DOUG	20

如果由下而上检查解决方案，就会发现先执行步骤 1 “找到比三个以上学生大的学生集”，如下所示（为了可读性，使用 DISTINCT 产生结果集的大小）：

```
select distinct s1.*
  from student s1,
       student s2,
       student s3,
       student s4
 where s1.age > s2.age
       and s2.age > s3.age
       and s3.age > s4.age
```

SNO	SNAME	AGE
2	CHUCK	21
5	STEVE	22
7	BRIAN	21
10	CHAD	21

如果对自联接感到困惑，那么只要集中了解 WHERE 子句。S1.AGE 比 S2.AGE 大，因此，

这时考虑了至少比一个学生大的学生。下一步，S2.AGE 比 S3.AGE 大，这时考虑了至少比两个学生大的学生。如果还是不明白，则尽力记住大于比较操作是可传递的。如果 S1.AGE 大于 S2.AGE，而且 S2.AGE 大于 S3.AGE，那么 S1.AGE 大于 S3.AGE。一旦理解了每一步返回的结果，那么降到自联接，并建立查询，这会很有帮助。例如，找到至少比一个学生大的所有学生（除了 JING 之外，应该返回所有学生）：

```
select distinct s1.*
  from student s1,
       student s2
 where s1.age > s2.age
```

SNO	SNAME	AGE
5	STEVE	22
7	BRIAN	21
10	CHAD	21
2	CHUCK	21
1	AARON	20
3	DOUG	20
9	GILLIAN	20
8	KAY	20
4	MAGGIE	19

下一步，找到至少比两个以上学生大的所有学生（现在，JING 和 MAGGIE 都应该从结果集中排除了）：

```
select distinct s1.*
  from student s1,
       student s2,
       student s3
 where s1.age > s2.age
       and s2.age > s3.age
```

SNO	SNAME	AGE
1	AARON	20
2	CHUCK	21
3	DOUG	20
5	STEVE	22
7	BRIAN	21
8	KAY	20
9	GILLIAN	20
10	CHAD	21

最后，找到至少比三个以上学生大的所有学生（结果集中只剩下了 CHUCK、STEVE、BRIAN 和 CHAD）：

```
select distinct s1.*
  from student s1,
       student s2,
       student s3,
       student s4
 where s1.age > s2.age
       and s2.age > s3.age
       and s3.age > s4.age
```

SNO	SNAME	AGE
2	CHUCK	21
5	STEVE	22
7	BRIAN	21



现在，知道了哪些学生比三个以上学生大，只要在一个子查询中用 NOT IN，返回除这四个学生之外的所有学生。

## B.4 回答有关“至少”的问题

“至多”的对立面是“至少”。通常，采用“至多”问题中描述的技巧变体，就可以解决“至少”问题。当解决“至少”问题时，把它们换一种说法“没有更少的”，会更有益。通常，如果在需求中能够识别阈值，那么就解决了一半问题。一旦知道了阈值，就可以决定采用一步（聚集函数和窗口函数，典型的是 COUNT）或两步（否定子查询）解决问题。

### 问题 6

找到至少选取两门课程的学生。

把问题重新描述为“找到选取两门以上课程的学生”或“找到选取了不少于两门课程的学生”，这是非常有帮助的。可以采用问题 4 中的技巧：使用聚集函数 COUNT 或窗口函数 COUNT OVER。最终结果集应该是：

SNO	SNAME	AGE
1	AARON	20
3	DOUG	20
4	MAGGIE	19
6	JING	18

### MySQL 和 PostgreSQL

使用聚集函数 COUNT，找到至少选取两门课程的学生：

```
1 select s.sno,s.sname,s.age
2   from student s, take t
3  where s.sno = t.sno
4  group by s.sno,s.sname,s.age
5 having count(*) >= 2
```

### DB2、Oracle 和 SQL Server

使用窗口函数 COUNT OVER，找到至少选取两门课程的学生：

```
1 select distinct sno,sname,age
2   from (
3 select s.sno,s.sname,s.age,
4        count(*) over (
5          partition by s.sno,s.sname,s.age
6        ) as cnt
7   from student s, take t
8  where s.sno = t.sno
9        ) x
10  where cnt >= 2
```

## 讨论

对于该解决方案的完整讨论，请参阅本节的问题4，采用的技巧与之相同。对于聚集解决方案，把表 STUDENT 联接到表 TAKE，而且在 HAVING 子句中使用 COUNT，以保留选取两门以上课程的学生。对于窗口解决方案，把表 STUDENT 联接到表 TAKE，对各分区进行计数，分区按表 STUDENT 的所有列定义。至此，只要保留 CNT 大于等于 2 的所有行。

### 原始解决方案

下面的解决方案对表 TAKE 使用自联接，以找到选取两门以上课程的学生。在子查询中，对 SNO 的等值联接，确保了每个学生都是对他自己的课程进行计算的。如果某个学生选取了一门以上的课程，那么 CNO 的大于比较操作将为真，否则 CNO 等于 CNO（仅有一门课程，只能与自己比较）。最后一步，返回属于子查询结果集的所有学生，如下所示：

```
select *
  from student
 where sno in (
select t1.sno
  from take t1,
       take t2
 where t1.sno = t2.sno
       and t1.cno > t2.cno
 )
```

SNO	SNAME	AGE
1	AARON	20
3	DOUG	20
4	MAGGIE	19
6	JING	18

## 问题 7

找到同时选取 CS112 和 CS114 的学生。学生也可能选取了其他课程，但必须选取 CS112 和 CS114。

这个问题与问题2相似，问题2查找选取两门以上课程的学生，而这里将查找至少选取两门课程的学生（只有 AARON 和 DOUG 同时选取了 CS112 和 CS114）。只要修改问题2中的解决方案，并应用于此。最终结果集应该是：

SNO	SNAME	AGE
1	AARON	20
3	DOUG	20

### MySQL 和 PostgreSQL

使用聚集函数 MIN 和 MAX，找到同时选取 CS112 和 CS114 的学生：

```
1 select s.sno, s.sname, s.age
2   from student s, take t
3  where s.sno = t.sno
```

```

4      and t.cno in ('CS114','CS112')
5      group by s.sno, s.sname, s.age
6      having min(t.cno) != max(t.cno)

```

## DB2、Oracle 和 SQL Server

使用窗口函数 MIN OVER 和 MAX OVER，找到同时选取 CS112 和 CS114 的学生：

```

1  select distinct sno, sname, age
2  from (
3  select s.sno, s.sname, s.age,
4         min(cno) over (partition by s.sno) as min_cno,
5         max(cno) over (partition by s.sno) as max_cno
6  from student s, take t
7  where s.sno = t.sno
8        and t.cno in ('CS114','CS112')
9        ) x
10 where min_cno != max_cno

```

## 讨论

两个解决方案都使用同一种技巧得到答案。IN 列表确保返回选取 CS112、CS114 或同时选取二者的学生。如果某个学生没有选取这两门课程，那么 MIN(CNO) 将等于 MAX(CNO)，从而会排除这个学生。为使这种操作更形象，下面给出了窗口解决方案的中间结果（为了更明确，添加了 T.CNO）：

```

select s.sno, s.sname, s.age, t.cno,
       min(cno) over (partition by s.sno) as min_cno,
       max(cno) over (partition by s.sno) as max_cno
from student s, take t
where s.sno = t.sno
      and t.cno in ('CS114','CS112')

```

SNO	SNAME	AGE	CNO	MIN_C	MAX_C
1	AARON	20	CS114	CS112	CS114
1	AARON	20	CS112	CS112	CS114
2	CHUCK	21	CS112	CS112	CS112
3	DOUG	20	CS114	CS112	CS114
3	DOUG	20	CS112	CS112	CS114
4	MAGGIE	19	CS112	CS112	CS112
6	JING	18	CS114	CS114	CS114

如果检验结果，会很容易明白，只有 AARON 和 DOUG 包含满足 MIN(CNO) != MAX(CNO) 条件的行。

## 原始解决方案

在 Rozenshtein 的原始解决方案中，针对表 TAKE 使用了自联接。下面给出了原始解决方案，如果索引恰当，该方案会非常高效：

```

select s.*
from student s,
     take t1,
     take t2
where s.sno = t1.sno
      and t1.sno = t2.sno
      and t1.cno = 'CS112'
      and t2.cno = 'CS114'

```

SNO	SNAME	AGE
1	AARON	20
3	DOUG	20

所有解决方案都确保：不管学生是否选取了其他课程，都必须选取 CS112 和 CS114。如果不能理解自联接，那么会发现理解下面的例子相当容易：

```
select s.*
  from take t1, student s
 where s.sno = t1.sno
    and t1.cno = 'CS114'
    and 'CS112' = any (select t2.cno
                       from take t2
                      where t1.sno = t2.sno
                        and t2.cno != 'CS114')
```

SNO	SNAME	AGE
1	AARON	20
3	DOUG	20

## 问题 8

找到至少比两个学生大的学生。

如果把问题描述为“找到比两个以上学生大的学生”，会更容易理解。可以使用问题 5 中的技巧。最终结果集如下所示（只有 JING 和 MAGGIE 不比两个以上学生大）：

SNO	SNAME	AGE
1	AARON	20
2	CHUCK	21
3	DOUG	20
5	STEVE	22
7	BRIAN	21
8	KAY	20
9	GILLIAN	20
10	CHAD	21

## MySQL 和 PostgreSQL

使用聚集函数 COUNT 和关联的子查询，找到至少比两个学生大的学生：

```
1 select s1.*
2   from student s1
3  where 2 <= ( select count(*)
4               from student s2
5              where s2.age < s1.age )
```

## DB2、Oracle 和 SQL Server

使用窗口函数 DENSE\_RANK，至少比两个学生大的学生：

```
1 select sno,sname,age
2   from (
3 select sno,sname,age,
4        dense_rank()over(order by age) as dr
5   from student
6   ) x
7  where dr >= 3
```

讨论

对于该解决方案的完整讨论，请参阅本节的问题5。两种解决方案采用的技巧都相同，唯一的差别是最后对计数或等级的计算方式不同。

原始解决方案

这个问题是问题6的变体，唯一差别就是处理STUDENT表的方式不同。问题6中的解决方案可以很容易地应用于“找到至少比两个学生大的学生”，如下所示：

```
select distinct s1.*
  from student s1,
       student s2,
       student s3
 where s1.age > s2.age
       and s2.age > s3.age
```

SNO	SNAME	AGE
1	AARON	20
2	CHUCK	21
3	DOUG	20
5	STEVE	22
7	BRIAN	21
8	KAY	20
9	GILLIAN	20
10	CHAD	21

B.5 回答有关“准确”的问题

读者可能认为，回答某件事是否为真的问题相当容易。在很多情况下，这都是非常容易的，但要回答某件事是否为“准确”真的问题，尤其是回答有关主/从数据联接的问题时，可能会有点儿棘手。问题来自“准确”的排他性。把它看作“唯一”可能会更有益。想一想，穿鞋的人和只穿鞋的人之间的差别，光满足条件是不够的，必须满足条件，而且确保不满足其他条件。

问题 9

找到正好只教一门课程的教授

换一种说法“找到只教一门课程的教授”，他们教的是哪门课程并不重要，关键是只教一门课程。最终结果集应该是：

LNAME	DEPT	SALARY	AGE
POMEL	SCIENCE	500	65

MySQL 和 PostgreSQL

使用聚集函数 COUNT，找到只教一门课程的教授：

```
1 select p.lname,p.dept,p.salary,p.age
2   from professor p, teach t
3  where p.lname = t.lname
```

```

4  group by p.lname,p.dept,p.salary,p.age
5  having count(*) = 1

```

## DB2、Oracle 和 SQL Server

使用窗口函数 COUNT OVER，找到只教一门课程的教授：

```

1  select lname, dept, salary, age
2    from (
3  select p.lname,p.dept,p.salary,p.age,
4         count(*) over (partition by p.lname) as cnt
5    from professor p, teach t
6   where p.lname = t.lname
7         ) x
8   where cnt = 1

```

## 讨论

把表 PROFESSOR 与表 TEACH 进行内联接，就确保删除了所有不教任何课程的教授。对于聚集解决方案，在 HAVING 子句中使用 COUNT，以返回只教一门课程的教授。窗口解决方案使用了 COUNT OVER 函数，但要注意，COUNT OVER 函数的 PARTITION 子句中使用了表 PROFESSOR 中的列，它们跟聚集解决方案的 GROUP BY 子句使用的列不同。在这个例子中，GROUP BY 和 PARTITION BY 子句不相同，这是很安全的，其原因是表 TEACHER 中的姓是唯一的，也就是说，分区中没有 P.DEPT、P.SALARY 和 PAGE 不会影响 COUNT 操作。在前一个解决方案中，窗口函数解决方案的 PARTITION 子句与聚集解决方案的 GROUP BY 子句故意使用了相同列，以表明 PARTITION 是可移动的，而且比 GROUP BY 更灵活。

## 原始解决方案

这个解决方案使用的技巧与问题 3 相同：进行两步操作找到答案。第一步，找到教两门以上课程的教授。第二步，找到只教一门课程，并且不属于步骤 1 结果集的所有教授。有关完整讨论，请参阅问题 3。该解决方案如下所示：

```

select p.*
  from professor p,
       teach t
 where p.lname = t.lname
    and p.lname not in (
select t1.lname
  from teach t1,
       teach t2
 where t1.lname = t2.lname
    and t1.cno > t2.cno
)

```

LNAME	DEPT	SALARY	AGE
-----	-----	-----	-----
POMEL	SCIENCE	500	65

## 问题 10

找到只选取 CS112 和 CS114 的学生（只选取了这两门课程，而没有选取其他课程的学生）。

但下面的查询返回一个空结果集：

```
select s.*
  from student s, take t
 where s.sno = t.sno
    and t.cno = 'CS112'
    and t.cno = 'CS114'
```

不存在这样的行，它的同一列包含两个不同值（假定用的是简单的标量数据类型，例如表 STUDENT 使用的数据类型），因此该查询永远不会有结果，Rozenshtein 的书针对编写查询时如何引起这种错误给出了解答。DOUG 是只选取 CS112 和 CS114 的唯一学生，因此应该是这个查询返回的唯一学生。

## MySQL 和 PostgreSQL

使用 CASE 表达式和聚集函数 COUNT，找到只选取 CS112 和 CS114 的学生：

```
1 select s.sno, s.sname, s.age
2   from student s, take t
3  where s.sno = t.sno
4  group by s.sno, s.sname, s.age
5 having count(*) = 2
6    and max(case when cno = 'CS112' then 1 else 0 end) +
7      max(case when cno = 'CS114' then 1 else 0 end) = 2
```

## DB2、Oracle 和 SQL Server

使用窗口函数 COUNT OVER 及 CASE 表达式，找到只选取 CS112 和 CS114 的学生：

```
1  select sno,sname,age
2    from (
3  select s.sno,
4         s.sname,
5         s.age,
6         count(*) over (partition by s.sno) as cnt,
7         sum(case when t.cno in ( 'CS112', 'CS114' )
8            then 1 else 0
9            end)
10        over (partition by s.sno) as both,
11        row_number()
12        over (partition by s.sno order by s.sno) as rn
13  from student s, take t
14  where s.sno = t.sno
15  ) x
16  where cnt = 2
17        and both = 2
18        and rn = 1
```

## 讨论

聚集解决方案采用的技巧与问题 1 和问题 2 的相同。表 STUDENT 与表 TAKE 的内联接确保排除了未选取任何课程的学生。HAVING 子句中的 COUNT 表达式保留只选取两门课程的学生。CASE 表达式用于计算课程数，把它的结果相加。对于只选取了 CS112 和 CS114 的学生，这个和是 2。

窗口解决方案采用的技巧与问题 1 和问题 2 的窗口解决方案相似。这个版本稍有不同，因

为CASE表达式将结果返回给窗口函数SUM OVER。这种解决方案的另一个变体是使用窗口函数ROW\_NUMBER，从而避免使用DISTINCT。没有进行最后筛选的窗口解决方案及其返回的结果如下所示：

```
select s.sno,
       s.sname,
       s.age,
       count(*) over (partition by s.sno) as cnt,
       sum(case when t.cno in ( 'CS112', 'CS114' )
              then 1 else 0
            end)
       over (partition by s.sno) as both,
       row_number()
       over (partition by s.sno order by s.sno) as rn
from student s, take t
where s.sno = t.sno
```

SNO	SNAME	AGE	CNT	BOTH	RN
1	AARON	20	3	2	1
1	AARON	20	3	2	2
1	AARON	20	3	2	3
2	CHUCK	21	1	1	1
3	DOUG	20	2	2	1
3	DOUG	20	2	2	2
4	MAGGIE	19	2	1	1
4	MAGGIE	19	2	1	2
5	STEVE	22	1	0	1
6	JING	18	2	1	1
6	JING	18	2	1	2

检验这些结果，可以看到，最终结果集包含 BOTH 和 CNT 都为 2 的行。RN 的值可以是 1，也可以是 2，都没关系，使用该列只是为了筛选掉重复行而不使用 DISTINCT。

### 原始解决方案

这种解决方案使用子查询及多个自联接，先找到至少选取三门课程的学生。下一步，对表 TAKE 使用自联接，找到选取了 CS112 和 CS114 的学生。最后一步，保留那些选取了 CS112 和 CS114、未选取三门以上课程的学生。该解决方案如下所示：

```
select s1.*
  from student s1,
       take t1,
       take t2
 where s1.sno = t1.sno
    and s1.sno = t2.sno
    and t1.cno = 'CS112'
    and t2.cno = 'CS114'
    and s1.sno not in (
select s2.sno
  from student s2,
       take t3,
       take t4,
       take t5
 where s2.sno = t3.sno
    and s2.sno = t4.sno
    and s2.sno = t5.sno
    and t3.cno > t4.cno
    and t4.cno > t5.cno
)
```



SNO	SNAME	AGE
3	DOUG	20

## 问题 11

找到只比两个学生大的学生。换一种说法“找到第三年轻的学生”。最终结果集应该是：

SNO	SNAME	AGE
1	AARON	20
3	DOUG	20
8	KAY	20
9	GILLIAN	20

## MySQL 和 PostgreSQL

使用聚集函数 COUNT 及关联子查询，找到第三年轻的学生：

```

1  select s1.*
2     from student s1
3     where 2 = ( select count(*)
4                  from student s2
5                  where s2.age < s1.age )

```

## DB2、Oracle 和 SQL Server

使用窗口函数 DENSE\_RANK，找到第三年轻的学生：

```

1  select sno,sname,age
2     from (
3  select sno,sname,age,
4         dense_rank()over(order by age) as dr
5     from student
6     ) x
7  where dr = 3

```

## 讨论

聚集解决方案使用标量子查询，找到比两个学生（只两个）大的所有学生。要查看这种方案的工作方式，可使用标量子查询重新编写该解决方案。在下面的例子中，列 CNT 表示比当前学生年轻的学生数：

```

select s1.*,
       (select count(*) from student s2
        where s2.age < s1.age) as cnt
  from student s1
 order by 4

```

SNO	SNAME	AGE	CNT
6	JING	18	0
4	MAGGIE	19	1
1	AARON	20	2
3	DOUG	20	2
8	KAY	20	2
9	GILLIAN	20	2
2	CHUCK	21	6
7	BRIAN	21	6
10	CHAD	21	6
5	STEVE	22	9

按照这种方式重新编写解决方案，使查看谁是第三年轻的学生（CNT值是2）更为容易。

使用窗口函数 DENSE\_RANK 的解决方案与标量子查询例子相似，每行都是基于比当前学生年轻的学生数分等级的（允许捆绑，而且没有缝隙）。下面的查询显示了 DENSE\_RANK 函数的输出：

```
select sno,sname,age,
       dense_rank()over(order by age) as dr
from student
```

SNO	SNAME	AGE	DR
6	JING	18	1
4	MAGGIE	19	2
1	AARON	20	3
3	DOUG	20	3
8	KAY	20	3
9	GILLIAN	20	3
2	CHUCK	21	4
7	BRIAN	21	4
10	CHAD	21	4
5	STEVE	22	5

最后一步，将查询包入内联视图，只保留 DR 为 3 的行。

### 原始解决方案

原始解决方案采用两个步骤：第一步，找到比三个以上学生大的学生；第二步，找到比两个学生大而且不属于步骤 1 结果集的学生。另外，Rozenshtein 把这个问题重新表述为“找到至少比两个学生大、至少不比三个学生大的学生。”该解决方案如下所示：

```
select s5.*
  from student s5,
       student s6,
       student s7
 where s5.age > s6.age
    and s6.age > s7.age
    and s5.sno not in (
select s1.sno
  from student s1,
       student s2,
       student s3,
       student s4
 where s1.age > s2.age
    and s2.age > s3.age
    and s3.age > s4.age
)
```

SNO	SNAME	AGE
1	AARON	20
3	DOUG	20
9	GILLIAN	20
8	KAY	20

上述解决方案使用的技巧与问题 5 相同。有关使用自联接的完整讨论，请参阅问题 5。

## B.6 回答有关“一些”或“所有”的问题

有关“一些”或“所有”的查询的典型要求是找到完全满足一个或多个条件的行。例如，如果查找吃所有蔬菜的人，实质上对要寻找的人而言，没有他们不吃的蔬菜。这种类型的问题表述一般归类为关系分割。有关“一些”的问题，密切注意问题的表达方式。考虑下面两个需求之间的差别：“选取任一门课程的学生”和“比任何火车都快飞机”。前一个描述的意思是“找到至少选取一门课程的学生”，而后面的描述意味着“找到比所有火车都快飞机。”

### 问题 12

找到选取所有课程的学生。

表 TAKE 中有关学生的课程数一定等于表 COURSES 中的课程总数。表 COURSES 中共有三门课程。AARON 选取了这三门课，他应该是返回的唯一学生。最终结果集应该是：

SNO	SNAME	AGE
1	AARON	20

### MySQL 和 PostgreSQL

使用聚集函数 COUNT，找到选取每门课程的学生：

```
1 select s.sno,s.sname,s.age
2   from student s, take t
3  where s.sno = t.sno
4  group by s.sno,s.sname,s.age
5 having count(t.cno) = (select count(*) from courses)
```

### DB2 和 SQL Server

使用窗口函数 COUNT OVER 及一个外联接，而不使用子查询：

```
1  select sno,sname,age
2    from (
3  select s.sno,s.sname,s.age,
4         count(t.cno)
5         over (partition by s.sno) as cnt,
6         count(distinct c.title) over() as total,
7         row_number() over
8         (partition by s.sno order by c.cno) as rn
9    from courses c
10   left join take t    on (c.cno = t.cno)
11   left join student s on (t.sno = s.sno)
12  ) x
13  where cnt = total
14        and rn = 1
```

### Oracle

Oracle9i 及更高版本的用户可以采用 DB2 解决方案。另外，也可以使用 Oracle 特有的外联接句法，对于 8i 或较早版本的用户，只能使用该方案：

```

1  select sno,sname,age
2      from (
3  select s.sno,s.sname,s.age,
4         count(t.cno)
5         over (partition by s.sno) as cnt,
6         count(distinct c.title) over() as total,
7         row_number() over
8         (partition by s.sno order by c.cno) as rn
9      from courses c, take t, student s
10     where c.cno = t.cno (+)
11           and t.sno = s.sno (+)
12           )
13     where cnt = total
14           and rn = 1

```

## 讨论

聚集解决方案使用子查询，返回课程总数；外层查询只保留那些课程数与子查询返回值相同的学生。窗口解决方案采用了另一种方法：它对表 COURSES 进行外联接，而不是使用子查询。窗口解决方案也用窗口函数返回学生选取的课程数（别名 CNT）以及表 COURSES 中的课程总数（别名 TOTAL）。下面的查询给出了这些窗口函数的中间结果：

```

select s.sno,s.sname,s.age,
       count(distinct t.cno)
       over (partition by s.sno) as cnt,
       count(distinct c.title) over() as total,
       row_number()
       over(partition by s.sno order by c.cno) as rn
from   courses c
       left join take t    on (c.cno = t.cno)
       left join student s on (t.sno = s.sno)
order by 1

```

SNO	SNAME	AGE	CNT	TOTAL	RN
1	AARON	20	3	3	1
1	AARON	20	3	3	2
1	AARON	20	3	3	3
2	CHUCK	21	1	3	1
3	DOUG	20	2	3	1
3	DOUG	20	2	3	2
4	MAGGIE	19	2	3	1
4	MAGGIE	19	2	3	2
5	STEVE	22	1	3	1
6	JING	18	2	3	1
6	JING	18	2	3	2

对于选取所有课程的学生，他的 CNT 应等于 TOTAL。使用 ROW\_NUMBER 可以从最终结果集筛选掉重复而不需要使用 DISTINCT。严格地讲，表 TAKE 和 STUDENT 的外联接并不是必需的，这是由于每门课程都至少被一名学生选取。如果存在一门学生不选取的课程，那么 CNT 就不等于 TOTAL，而且会返回 SNO、SNAME 和 AGE 都为 NULL 值的行。下面的例子创建了一个新课程，没有学生选取它。下面的查询示意了如果不存在学生选取的课程，其中间结果集的情况（为了更清楚，下面也引入了 C.TITLE）：

```

insert into courses values ('CS115','BIOLOGY',4)
select s.sno,s.sname,s.age,c.title,
       count(distinct t.cno)
       over (partition by s.sno) as cnt,
       count(distinct c.title) over() as total,

```

```

    row_number()
  over(partition by s.sno order by c.cno) as rn
from courses c
  left join take t    on (c.cno = t.cno)
  left join student s on (t.sno = s.sno)
order by 1

```

SNO	SNAME	AGE	TITLE	CNT	TOTAL	RN
1	AARON	20	PHYSICS	3	4	1
1	AARON	20	CALCULUS	3	4	2
1	AARON	20	HISTORY	3	4	3
2	CHUCK	21	PHYSICS	1	4	1
3	DOUG	20	PHYSICS	2	4	1
3	DOUG	20	HISTORY	2	4	2
4	MAGGIE	19	PHYSICS	2	4	1
4	MAGGIE	19	CALCULUS	2	4	2
5	STEVE	22	CALCULUS	1	4	1
6	JING	18	CALCULUS	2	4	1
6	JING	18	HISTORY	2	4	2
			BIOLOGY	0	4	1

检验这些结果，很容易会看到，加上最终筛选条件后，就不会返回任何行。另外，要记住：窗口函数将在 WHERE 子句计算之后开始运行，所以，在计算表 COURSES 中可用的课程总数时，必须使用 DISTINCT（否则，结果集中得到的总数并不是所有学生选取的课程总数，即 `select count(cno) from take`）。

注意：这个例子中表 TAKE 所用的样品数据没有重复，因此该解决方案能够正常运行。如果 TAKE 中存在重复，例如，某个学生三次选取了同一门课程，这个解决方案就会失败。在该解决方案中，要处理重复也是非常简单的，只需在执行 T.CNO 计数时添加 DISTINCT 即可，这样，该解决方案就会正常运行。

## 原始解决方案

原始解决方案以一种非常灵巧的方式使用笛卡儿积，从而避免使用聚集。下面的查询是以原始解决方案为基础的：

```

select *
  from student
 where sno not in
    ( select s.sno
      from student s, courses c
      where (s.sno,c.cno) not in (select sno,cno from take) )

```

Rozenshtein 把问题重新描述为“哪些学生不属于未选取一门课程的学生？如果按这种方式看问题，那么现在需要处理有关“否定”的问题。回忆一下，Rozenshtein 是如何处理“否定”的：

记住，真正的“否定”需要两个步骤：要找到“谁不在”，应该先找到“谁在”，然后去除他们。

最里层的子查询返回所有有效的 SNO/CNO 组合；中间层的子查询使用了表 STUDENT 和 COURSES 之间的笛卡儿积，返回所有学生和所有课程的组合（也即选取每门课程的每个

学生), 并筛选掉有效的 SNO/CNO 组合 (只留下“虚构的” SNO/CNO 组合); 最外层的查询从表 STUDENT 中返回的行满足下列条件: SNO 不属于中间子查询返回的值。下面的查询可能会使解决方案更清晰。为了具有可读性, 这里只使用了 AARON 和 CHUCK (只有 AARON 选取了所有课程):

```
select *
  from student
 where sno in ( 1,2 )
```

SNO	SNAME	AGE
1	AARON	20
2	CHUCK	21

```
select *
  from take
 where sno in ( 1,2 )
```

SNO	CNO
1	CS112
1	CS113
1	CS114
2	CS112

```
select s.sno, c.cno
  from student s, courses c
 where s.sno in ( 1,2 )
 order by 1
```

SNO	CNO
1	CS112
1	CS113
1	CS114
2	CS112
2	CS113
2	CS114

这些查询分别显示了表 STUDENT 中有关 AARON 和 CHUCK 的行、AARON 和 CHUCK 选取的课程以及返回 AARON 和 CHUCK 选取所有课程的笛卡儿积。AARON 的笛卡儿积的结果集与表 TAKE 中为 AARON 返回的结果集相匹配, 但 CHUCK 的笛卡儿积的结果包含两个“虚构”行, 这与表 TAKE 中他的对应行不匹配。下面的查询是中间子查询, 它使用 NOT IN 筛选掉有效的 SNO/CNO 组合:

```
select s.sno, c.cno
  from student s, courses c
 where s.sno in ( 1,2 )
       and (s.sno,c.cno) not in (select sno,cno from take)
```

SNO	CNO
2	CS113
2	CS114

注意, 中间子查询并不返回 AARON (因为 AARON 选取了所有课程)。中间子查询的结果集中只是由笛卡儿积创造的行, 并不是说 CHUCK 实际上选取了这些课程。最外层的查询返回表 STUDENT 中满足下列条件的行, 即它的 SNO 不属于中间子查询返回的 SNO:

```
select *
  from student
 where sno in ( 1,2 )
    and sno not in
      (select s.sno from student s, courses c
       where s.sno in ( 1,2 )
          and (s.sno,c.cno) not in (select sno,cno from take))
```

SNO	SNAME	AGE
1	AARON	20

问题 13

找到比其他所有学生都大的学生。

换一种说法“找到年龄最大的学生”。最终结果集应该是：

SNO	SNAME	AGE
5	STEVE	22

MySQL 和 PostgreSQL

在子查询中使用聚集函数 MAX，找到年龄最大的学生：

```
1 select *
2   from student
3  where age = (select max(age) from student)
```

DB2、Oracle 和 SQL Server

在内联视图中使用窗口函数 MAX OVER，找到年龄最大的学生：

```
1 select sno,sname,age
2   from (
3 select s.*,
4        max(s.age)over() as oldest
5   from student s
6  ) x
7  where age = oldest
```

讨论

两种解决方案都使用函数 MAX，查找年龄最大的学生。子查询解决方案先找到表 STUDENT 中的最大年龄，并把它返回给外层查询，该查询将找到这个年龄的学生。窗口版本的解决方案与子查询的方案相同。只是该解决方案在每行中都返回的最大年龄。窗口查询的中间结果如下所示：

```
select s.*,
       max(s.age) over() as oldest
  from student s
```

SNO	SNAME	AGE	OLDEST
1	AARON	20	22
2	CHUCK	21	22
3	DOUG	20	22
4	MAGGIE	19	22

5	STEVE	22	22
6	JING	18	22
7	BRIAN	21	22
8	KAY	20	22
9	GILLIAN	20	22
10	CHAD	21	22

要找到年龄最大的学生，只需要保留满足 AGE = OLDEST 条件的行。

### 原始解决方案

原始解决方案在子查询中对表 STUDENT 使用了自联接，以找到比某些学生小的所有学生。外层查询返回了表 STUDENT 中满足下列条件的所有学生：他们不属于子查询返回的结果集。可以把这种操作描述为“找到满足下述条件的所有学生：他们不属于至少比一个学生小的学生”：

```
select *
  from student
 where age not in (select a.age
                   from student a, student b
                   where a.age < b.age)
```

子查询使用笛卡儿积，以便找到 A 中 B 中的所有年龄都小年龄。只有不比其他年龄小的年龄才是最大年龄，最大年龄不是由子查询返回的。外层查询使用 NOT IN，返回表 STUDENT 中的所有 AGE 不属于子查询返回的 AGE 的行（如果返回了 A.AGE，就意味着表 STUDENT 中有一个 AGE 比它大）。如果不能理解这些内容，则查看下面的查询。从概念上讲，它们的工作方式相同，但下面的查询可能更常见一些：

```
select *
  from student
 where age >= all (select age from student)
```



## 作者简介

---

**Anthony Molinaro** 是 Wireless Generation 公司的数据库开发人员。他多年从事帮助开发人员改进其 SQL 查询的工作，具有丰富的实践经验。Anthony 酷爱 SQL，在相关领域，他小有名气，客户在遇到困难的 SQL 查询问题时，就会想到他，他总能起到关键作用。他博学多才，对关系理论有深入的理解，有 9 年解决复杂 SQL 问题的实战经验。Anthony 通晓新的和功能强大的 SQL 功能，比如，添加到最新 SQL 标准中的窗口函数语法等。

## 封面介绍

---

封面上的动物是飞龙科蜥蜴，这种蜥蜴属于飞龙科家族，有 300 多个类别。飞龙科蜥蜴生存在非洲、亚洲、澳大利亚以及南部欧洲，它们的特点是，腿粗壮有力，有些种类还可以变色。与其他蜥蜴物种不同的是，飞龙科蜥蜴如果丢了尾巴，没有再生能力。它们的适应能力强，能生存在迥然不同的环境中，从酷热难耐的热带沙漠，到温暖湿润的热带雨林，都有它们的身形足迹。

飞龙科蜥蜴有些种类还是人们的宠物呢，比如鬃蜥（鬃狮蜥物种）。它们性格安静，样子有些古怪，个头只有 20 英寸大小。但即便是个子小，它们也算是“巨大的”蜥蜴了，因而需要有充足的空间。雄性通常会有自己的领地，并竭力维护使其不受侵犯，而且，尽管它们是社会性的动物，但过度的密集拥挤也会带来生存压力，尤其是在它们没有躲藏空间的时候。过度拥挤会导致争斗，因而可能会受伤，比如失去脚趾或尾巴，以及失去食欲。

鬃蜥的头呈三角状，下巴上伸出许许多多的尖头。这些尖头就像胡须一样，也算得名副其实了。这些鬃须长在边上。它们打开大嘴的时候，就会露出尖尖的鬃须，以此来吓唬其他食肉动物以及其他鬃蜥。它们还会伸展身体，使自己看起来更大些。鬃蜥作为宠物，如果主人性格温和，它的栖息地比较舒适，那么，它就不会展示自己的鬃须。

尽管鬃蜥产自澳大利亚，但是，是美国商人从欧洲进口鬃蜥后，进行大量繁殖，然后销售世界各地。原因是澳大利亚对于野生动物有严格的出口法规。

飞蜥是飞龙科蜥蜴的另一个变种。它的个头稍小于12英寸，身体长长的，细细的，肋骨处有副翼。雄性飞蜥会占有两三棵树作为它的领土，每棵树上，生活着1~3个雌性蜥蜴。为了移动位置，它会从一棵树上，或者是高高的位置，像翅膀一样展开副翼，进行滑翔。然而，在刮风下雨的时候，它不会飞翔。在受到威胁的时候，飞蜥会展开翅膀，这样会显得更强大。

飞龙科蜥蜴还有一个有趣的种类，就是红头鬣蜥（彩虹飞蜥），它们生活在非洲的撒哈拉沙漠。它们群居生活，每个群体有10~20个蜥蜴，年长的雄性蜥蜴是它们的“首领”。夜深人静的时候，它们的身体会变成深棕色；拂晓时分，身体变成浅蓝色，头和尾巴变成浅橙色。它们的皮肤颜色会随心情而变化，可以通过颜色，知道它们的情绪。比如说，雄性蜥蜴争斗的时候，他们的头会变为棕色，身上布满白色斑点。

